R News

The Newsletter of the R Project — Volume 2/3, December 2002

Editorial

by Kurt Hornik

Time to say good-bye ... with the new system of rolling three-year periods of office for the members of the Editorial Board, my term comes to an end on Dec 31, with Doug Bates joining the board in 2003, and Fritz Leisch taking over as Editor-in-Chief.

Having started *R News* from scratch together with Fritz, I am of course very content with what we have achieved thus far. And that does not change when comparing our initial intentions, as stated in the editorial of the first (January 2001) issue of *R News*, to the present state. Both quality and quantity of the material published in the past two years confirm the success of this newsletter in filling the gap between mailing lists and scientific journals. (The first issue also mentions the idea of having a regular column on "Applications". Guess that's what's usually called "work in progress.")

In serving its role as one of the key information resources of the R community, *R News* of course needs to do more that "just" provide news from the most recent R releases and articles introducing packages available from CRAN or other R package repositories—there should be information on books and events related to R, a programmer's niche, a help desk ... and if possible, on a regular basis. Thus, we are very happy that Uwe Ligges, known to many as a key provider of answers on the r-help mailing list, has agreed to edit a regular "Help Desk" column, starting in this issue with an article on "Automation of Mathematical Annotation in Plots". Welcome aboard, Uwe!

Also in this issue, Angelo Canty continues the series on primers for recommended packages with "Resampling Methods in R: The boot Package". In "Mixing R and LaTeX", Fritz Leisch starts a mini-series on Sweave, an exciting flexible framework for mixing text with R code for automatic document generation. Steffen Lauritzen introduces a new, R-based project on "gRaphical Modeling in R". And of course, there is much more ...

Best wishes to everyone for 2003.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

Contents of this issue:

Resampling Methods in R: The boot Package

by Angelo J. Canty

Introduction

The bootstrap and related resampling methods are statistical techniques which can be used in place of standard approximations for statistical inference. The basic methods are very easily implemented but for the methods to gain widespread acceptance among users it is necessary that they be implemented in standard statistical packages. In this article I will describe the **boot** package which implements many variants of resampling methods in R. The package was originally written as an S-Plus library released in conjunction with the book by Davison and Hinkley (1997). Subsequently the library was ported to R by Brian Ripley. The **boot** package described here is distinct from the limited suite of bootstrap functions which are now included in S-Plus. It is also distinct from the **bootstrap** package originally written by R. Tibshirani for Efron and Tibshirani (1993).

In this paper I describe the main components of the **boot** package and their use. I will not go into any theoretical detail about the methods described. It is strongly recommended, however, that the user read Davison and Hinkley (1997) before using the package. Initially I will describe the basic function for bootstrapping i.i.d. data and analyzing the bootstrap output. Then I will describe functions for sampling randomly right-censored data and time series data. The availability of the bootstrap for such non-standard cases is one of the major advantages of the **boot** package.

The main bootstrap function

The most important function in the package is the `boot` function which implements resampling methods for i.i.d. data. The basic bootstrap in such cases works by fitting a distribution function $\hat{F}$ to the unknown population distribution F. The Monte Carlo bootstrap method then proceeds by taking R samples, of the same size as the original sample, from $\hat{F}$. In the parametric bootstrap $F = F_\psi$ is a member of a class of distribution functions indexed by the parameter vector ψ and so $\hat{F} = F_{\hat{\psi}}$ where $\hat{\psi}$ is some consistent estimate of ψ. It is then the responsibility of the user to supply (through the parameter `ran.gen`) a function which takes the original data and returns a sample from $\hat{F}$. In the nonparametric case the estimate of F is the empirical distribution function. In this case a sample from $\hat{F}$ can be found by sampling with replacement from the original data, $X_1, \ldots, X_n$. In `boot` all R samples are found by constructing an $R \times n$ matrix of random integers from `1:n`. Indexing the original data by each row of this matrix gives a bootstrap sample.

Let us now suppose that interest is in some functional $\theta = t(F)$. The *plug-in* estimate of this functional is then $t = t(\hat{F})$. Corresponding to a bootstrap sample is a distribution function $\hat{F}^*$. We can then use $t^* = t(\hat{F}^*)$ to estimate t. Under suitable regularity conditions we can then approximate the distribution of $t - \theta$ by the empirical distribution of $t^* - t$. Thus estimates of the bias and variance of the estimator T are

$$b = \bar{t}^* - t, \quad v = \frac{1}{R-1}\sum_{r=1}^{R}(t_r^* - \bar{t}^*)^2 \qquad (1)$$

Since `boot` is designed to be a general function for the bootstrap, the user must supply a function (`statistic`) which calculates the required functional. In the parametric bootstrap this is simply a function of a dataset. In the non-parametric case, `statistic` must be a function of the original dataset and a second argument which is used to determine a bootstrap sample. The simplest form that this second argument can take is to be a vector of indices such as a row of the index matrix constructed by `boot`. An equivalent method for exchangeable data is to supply the vector of frequencies of the original data points in the bootstrap sample. Another alternative is to supply the probability weights corresponding to $\hat{F}^*$. For bootstrap samples such weights are simply the frequencies divided by the sample size. They have the advantage, however, that any set of n weights which sum to 1 can be used. This allows for numerical differentiation of the functional $t(\cdot)$ at the datapoints which gives us the *empirical influence values*. Note that `boot` automatically normalizes weights to sum to 1 prior to calling `statistic`. The user must tell `boot` which second argument is being expected by `statistic`, this is achieved by specifying the `stype` parameter which can be `"i"`, `"f"` or `"w"`.

The following is an example of using `boot` at its most basic level. For many uses, this will be sufficient to run the bootstrap. The code performs a nonparametric bootstrap for the mean of the `aircondit` data.

```
> mean.w <- function(x, w) sum(x*w)
> air.boot <- boot(data=aircondit$hours,
+                  statistic=mean.w,
+                  R=999, stype="w")
```

The use of this second argument may seem confusing at first but it allows more flexibility in the types of data structures that we can bootstrap and how bootstrapping is applied to the data. For example, consider the case of bootstrapping for linear models. The data would generally be a matrix or dataframe. The two methods that could be used are to resample rows or resample residuals and then reconstruct a response vector. The following code shows how both of these can be achieved for the catsM data set.

```
> data(catsM)
> cats.lm <-lm(Hwt~Bwt, data=catsM)
> cats1 <- catsM
> cats1$fit <- fitted(cats.lm)
> cats1$res <- resid(cats.lm)
> cats.fit <- function(data) {
+    mod <- lm(data$Hwt~data$Bwt)
+    c(coef(mod),
+      summary(mod)$coef[,2]^2) }
> case.fun <- function(d,i)
+    cats.fit(d[i,])
> model.fun <- function(d,i) {
+    d$Hwt <- d$fit+d$res[i]
+    cats.fit(d) }
> cats.case <- boot(cats1, case.fun,
+                   R=999)
> cats.mod <- boot(cats1, model.fun,
+                  R=999)
```

One advantage of the **boot** package is that it implements many variants on the basic non-parametric bootstrap method. One obvious extension is to multi-sample problems. The user need only specify strata as a numeric vector or factor defining the groups. This fits different empirical distribution functions to each stratum and samples accordingly. Another possibility is that we may want to resample from the data with unequal weights. This arises in the context of bootstrap hypothesis testing and in using importance sampling with the bootstrap as suggested by Johns (1988) and Davison (1988). The sampling probabilities are passed using the weights argument to boot. Other types of resampling can also be done and are specified using the sim argument. Two of these are attempts at more efficient Monte Carlo sampling for the bootstrap; the balanced bootstrap (Davison et al., 1986) and the antithetic bootstrap (Hall, 1989). The final option is to resample without replacement as required for permutation tests.

Analysis of bootstrap output

The result of calling the boot function is that an object having the class "boot" is returned. This object contains most of the inputs to the boot function or the default values of those not specified. It also contains three additional components. The first of these is t0 which is the result of evaluating the statistic on the original dataset. The second, t, is the matrix of bootstrap replicates. Each row of the matrix corresponds to the value of the statistic applied to a bootstrap dataset. Finally, there is a component seed which contains the value of .Random.seed used to start the Monte Carlo sampling. There are two main reasons why this component is useful. The first is the issue of reproducibility of the bootstrap. In research it is often useful to apply different statistics to the same set of bootstrap samples for comparisons. Without the saved random seed this would not be possible. The second reason is that there are situations in which it is important to be able to look at the bootstrap samples themselves. This can be done by constructing the matrix of bootstrap indices (or frequencies). In an early version of the package that matrix was stored but that required excessive storage. By storing the random seed we can recreate the matrix whenever required in very little time. The function boot.array does this.

The two main methods for boot objects are print and plot methods. If the user prints a boot object then a short summary of the bootstrap results are given. Here are the results of case resampling the catsM dataset.

```
> cats.case
ORDINARY NONPARAMETRIC BOOTSTRAP

Call:
boot(data = cats1, statistic = case.fun,
     R = 999)

Bootstrap Statistics :
    original     bias    std. error
t1*  -1.1841  0.029573     1.14684
t2*   4.3127 -0.010809     0.40831
t3*   0.9966 -0.012316     0.16043
t4*   0.1155 -0.001534     0.01769
```

In this example, the statistic returns a vector of length 4, the first two components are the coefficient estimates and the second two are the estimated variances of the estimates from the usual linear model theory. We note that the bootstrap standard errors are 15–20% higher than the usual standard errors.

It is not safe to use the output of a bootstrap without first looking at a graphical plot of the bootstrap replicates. They are the first and most basic check that the bootstrap has produced sensible results. One common problem is that of discreteness of the bootstrap distribution. If this is a problem it should be fairly evident from the plots. The solution may be as simple as increasing the number of replicates or there may be some fundamental problem such as occurs in the case of the sample maximum. In either case the plots will alert the user that the results of this bootstrap cannot be used for inference. The plots may also show up bugs in the coding of the statistic which were not previously evident. Because of the nature of a bootstrap sample, it is possible to get datasets not normally seen in practice (such as having many ties) and the code which worked for the original sample

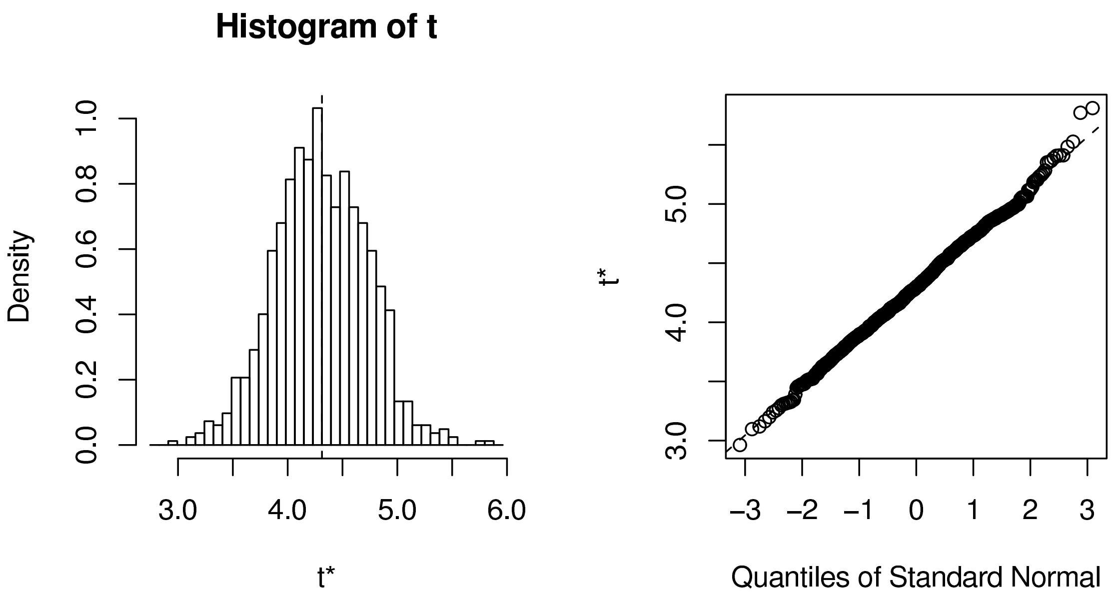

Figure 1: Plots of the bootstrap output for the slope of the `catsM` dataset using case resampling. The command used was `plot(cats.case, index=2)`.

may not work for some bootstrap samples. Figure 1 shows the plots for the slope estimate of the `catsM` dataset using case resampling.

A more sophisticated diagnostic tool is the *Jackknife-after-bootstrap* plot suggested by Efron (1992). The `plot` method may call the function `jack.after.boot` to draw these. Users should note that the plots produced by `jack.after.boot` are very different from those produced by the S-Plus function `plot.jack.after.bootstrap`. As a diagnostic, I think that the version I have used are more informative.

Having examined the bootstrap output and determined that it seems to be sensible the user can proceed to use the output. The most common use is to producing confidence intervals. The function `boot.ci` can be used for this. This takes a bootstrap output object and returns one or more types of confidence interval for a scalar component of the functional being estimated. There is an `index` argument to specify which component should be analyzed. `boot.ci` can produce five different types of bootstrap confidence interval. The *bootstrap normal* interval assumes an asymptotic normal distribution an used the bootstrap estimates of bias and variance for the parameters of the distribution. The *basic bootstrap* and *bootstrap percentile* intervals have less restrictive assumptions. All three intervals are asymptotically equivalent but the latter two tend to have better small sample properties when the normal assumption is questionable. This can be checked using the normal quantile plot produced by the `plot` method. Two intervals that are asymptotically better than these are the *studentized* and *BCa* intervals. The price for this improvement is more calculation. For the studentized bootstrap one needs a consistent estimate of the variance of each bootstrap replicate. Such variances can be found using asymptotic considerations or through methods such as the jackknife or non-parametric delta method (infinitesimal jackknife). Another alternative is to use a nested bootstrap but this can rapidly become computationally prohibitive. Whichever method is chosen, it must be done for each bootstrap sample. The BCa interval requires calculation of the empirical influence values for the original sample only so the extra computation is minimal but the statistic should be in weighted form unless a closed form is readily available. See Canty et al. (1996) for a simulation study comparing the various intervals

Resampling censored data

In many practical settings, data is censored and so the usual bootstrap is not applicable. The function `censboot` implements the bootstrap for random right-censored data. Such data is typically comprised of the bivariate observations (Y_i, D_i) where

$$Y_i = \min(X_i, C_i) \qquad D_i = I(X_i \le C_i)$$

where $X_i \sim F$ and $C_i \sim G$ independently and $I(A)$ is the indicator function of the event A. Nonparametric estimates of F and G are given by the Kaplan–Meier estimates $\hat{F}$ and $\hat{G}$, the latter being obtained by replacing d_i by $1 - d_i$. We can then proceed by sampling $X_1^*, \ldots, X_n^*$ from $\hat{F}$ and independently sampling $C_1^*, \ldots, C_n^*$ from $\hat{G}$. (Y_i^*, D_i^*) can then be found from (X_i^*, C_i^*) in the same way as for the original data. Efron (1981) showed that this is identical to resampling with replacement from the original pairs.

An alternative approach is the conditional bootstrap. This approach conditions the resampling on the observed censoring pattern since this is, in effect, an ancillary statistic. We therefore sample $X_1^*, \ldots, X_n^*$ from $\hat{F}$ as before. If the i^{th} observation is censored then we set $C_i^* = y_i$ and if it is not censored we sample an observation from the estimated conditional distribution of C_i given that $C_i > y_i$. Having thus obtained $X_1^*, \ldots, X_n^*$ and $C_1^*, \ldots, C_n^*$ we proceed as before. There is one technicality which must be addressed for this method to proceed. Suppose that the maximum value of $y_1, \ldots, y_n$, y_k say, is a censored observation. Then $X_k^* < y_k$ and $C_k^* = y_k$ so the bootstrap observation will always be uncensored. Alternatively, if the the maximum value is uncensored, the estimated conditional distribution does not exist. In order to overcome these problems we add one extra point to the dataset which has an observed value greater than $\max(y_1, \ldots, y_n)$ and has the opposite value of the censoring indicator to the maximum.

One other method for resampling from censored data is the weird bootstrap introduced by Andersen et al. (1993). This works by simulating from the Nelson-Aalen estimate of the cumulative hazard function.

The function `censboot` implements all of these resampling schemes. The user simply specifies the data (as a matrix or data.frame with at least 2 columns), the statistic (an R function taking the dataset as its input), the number of resamples and the Kaplan–Meier estimates of the survival and censoring distributions as found using the function `survfit` in the recommended package **survival**.

In practice there are usually covariates which affect the survival distribution. The most common assumption is that the survival distribution depends on the covariates, x, through a *Cox Proportional Hazards Model* (Cox, 1972).

$$1 - F(y; \beta, x) = \{1 - F_0(y)\}^{\exp(x^T \beta)}$$

where β is a vector of unknown parameters and $1 - F_0(y)$ is a baseline survivor function. The function `coxph` in the **survival** package fits such models. For the bootstrap this does not cause a great complication. The only difference from before is the estimated distribution from which failure times are generated. Earlier we mentioned that, without covariates, sampling from the model was identical to sampling pairs, this is no longer true in the case with covariates. Both methods are possible with `censboot`, specifying `sim="ordinary"` will always sample cases whereas `sim="model"` will resample from the Cox model if appropriate and otherwise will resample cases.

The following example looks at the ulcerated cases in the `melanoma` dataset. The aim is to produce a confidence interval for the exponent of the coefficient of tumor thickness in the proportional hazards model. Note that it is necessary to ensure that the censoring indicator is of the form specified above.

```
> data(melanoma)
> library(survival)
> mel <- melanoma[melanoma$ulcer==1,]
> mel$cens <- 1*(mel1$status==1)
> mel.cox <- coxph(Surv(time, cens)
+                  ~thickness, data=mel)
> mel.surv <- survfit(mel1.cox)
> mel.cens <- survfit(Surv(time,1-cens)
+                     ~1, data=mel)
> mel.fun <- function(d) {
+    coxph(Surv(time, cens)~thickness,
+          data=d)$coefficients }
> mel.boot <- censboot(mel, mel.fun,
+                R=999, sim="cond",
+                F.surv=mel.surv,
+                G.surv=mel.cens,
+                cox=mel.cox,
                 index=c(1,8))
> mel.boot
CONDITIONAL BOOTSTRAP FOR CENSORED DATA
Call:
censboot(data=mel, statistic=mel.fun,
         R=999, F.surv=mel.surv,
         G.surv=mel.cens, sim="cond",
         cox=mel.cox, index=c(1,8))

Bootstrap Statistics :
      original      bias    std. error
t1* 0.09971374 0.03175672   0.0456912

>  boot.ci(mel.boot,
+           type=c("basic", "perc"),
+           h=exp)
BOOTSTRAP CONFIDENCE INTERVAL
CALCULATIONS
Based on 999 bootstrap replicates

CALL :
boot.ci(boot.out=mel.boot,
        type=c("basic", "perc"),  h=exp)

Intervals :
Level     Basic         Percentile
95%   (0.952, 1.158)  (1.052, 1.258)
Calculations and Intervals on
Transformed Scale
```

The usual interval from `summary(mel.cox)` is $(1.0205, 1.1962)$ which is narrower than the bootstrap intervals suggesting that the asymptotic interval may be undercovering.

Resampling time series

In applying the bootstrap to time series data it is essential that the autocorrelations be properly accounted for. The most common way of doing this is to sample the observations in blocks rather than individually (Künsch, 1989). Thus if we choose a block length l and we have $n = ml$ for some integer m then the resampled time series is constructed by putting m blocks together. When $m = n/l$ is not an integer the last block is shortened so that the resampled time series is of the appropriate length. Blocks are usually taken as overlapping so that there are $n - l + 1$ possible blocks. This can be increased to n by allowing blocks to wrap around from the end to the start of the time series. One problem with the block bootstrap is that the resulting time series is not stationary. Politis and Romano (1994) proposed the *stationary bootstrap* to overcome this problem. In the stationary bootstrap the block length is randomly generated with a geometric distribution and the block start is selected randomly from the integers $\{1, \ldots, n\}$. Wrapping at the end of the time series is necessary for this method to work correctly. `tsboot` can do either of these methods by specifying `sim="fixed"` or `sim="geom"` respectively. A simple call to `tsboot` includes the time series, a function for the statistic (the first argument of this function being the time series itself), the number of bootstrap replicates, the simulation type and the (mean) block length .

```
> library(ts)
> data(lynx)
> lynx.fun <- function(tsb) {
+     fit <- ar(tsb, order.max=25)
+     c(fit$order, mean(tsb)) }
> tsboot(log(lynx), lynx.fun, R=999
+        sim="geom", l=20)
STATIONARY BOOTSTRAP FOR TIME SERIES
Average Block Length of 20

Call:
tsboot(tseries=log(lynx),
       statistic=lynx.fun,
       R=999, l=20, sim="geom")

Bootstrap Statistics :
     original        bias    std. error
t1* 11.000000 -6.593593594   2.5400596
t2*  6.685933 -0.001994561   0.1163937
```

An alternative to the block bootstrap is to use model based resampling. In this case a model is fitted to the time series so that the errors are i.i.d. The observed residuals are sampled as an i.i.d. series and then a bootstrap time series is reconstructed. In constructing the bootstrap time series from the residuals, it is recommended to generate a long time series and then discard the initial burn-in stage. Since the length of burn-in required is problem specific, `tsboot` does not actually do the resampling. Instead the user should give a function which will return the bootstrap time series. This function should take three arguments, the time series as supplied to `tsboot`, a value `n.sim` which is the length of the time series required and the third argument containing any other information needed by the random generation function such as coefficient estimates. When the random generation function is called it will be passed the arguments `data`, `n.sim` and `ran.args` passed to `tsboot` or their defaults.

One problem with the model-based bootstrap is that it is critically dependent on the correct model being fitted to the data. Davison and Hinkley (1997) suggest *post-blackening* as a compromise between the block bootstrap and the model-based bootstrap. In this method a simple model is fitted and the residuals are found. These residuals are passed as the dataset to `tsboot` and are resampled using the block (or stationary) bootstrap. To create the bootstrap time series the resampled residuals should be put back through the fitted model filter. The function `ran.gen` can be used to do this.

```
> lynx1 <- log(lynx)
> lynx.ar <- ar(lynx1)
> lynx.res <- lynx.ar$resid
> lynx.res <- [!is.na(lynx.res)]
> lynx.res <- lynx.res-mean(lynx.res)
> lynx.ord <- c(lynx.ar$order,0,0)
> lynx.mod <- list(order=lynx.ord,
+                  ar=lynx.ar$ar)
> lynx.args <- list(mean=mean(lynx1),
+                   model=lynx.mod)
> lynx.black <- function(res, n.sim,
+                        ran.args) {
+    m <- ran.args$mean
+    ts.mod <- ran.args$model
+    m+filter(res, ts.mod$ar,
+             method="recursive") }
> tsboot(lynx.res, lynx.fun, R=999,
+       l=20, sim="fixed", n.sim=114,
+       ran.gen=lynx.black,
+       ran.args=lynx.args)
POST-BLACKENED BLOCK BOOTSTRAP FOR
TIME SERIES
Fixed Block Length of 20

Call:
tsboot(tseries=lynx.res,
       statistic=lynx.fun, R=999,
       l=20, sim="fixed", n.sim=114,
       ran.gen=lynx.black,
       ran.args=lynx.args)

Bootstrap Statistics :
         original    bias    std. error
t1*  0.000000e+00 9.732733  3.48395113
t2* -4.244178e-18 6.685819  0.09757974
```

A final method which is available for bootstrapping of time series is *phase scrambling*. Unlike the other methods described above, phase scrambling works on the frequency domain. See Braun and Kulperger (1997) for a discussion of the properties of this method.

Further comments

In this article I have attempted to describe concisely the main functions in the **boot** package for bootstrapping. The package also has functions which implement saddlepoint approximations to the bootstrap as described in Canty and Davison (1999). There are also functions which do exponential tilting of the resampling distribution and other forms of importance resampling in the bootstrap. These are quite specialized uses of the package and so the user is advised to read the relevant sections of Davison and Hinkley (1997) before using these functions.

Acknowledgments

I would like to thank A. C. Davison and V. Ventura for their many helpful suggestions in the development of this library. Thanks are also due to B. D. Ripley for a great deal of help and for porting the code to R. Any bugs in the code, however, are my responsibility and should be reported to me in the first instance.

Bibliography

Andersen, P. K., Borgan, Ø., Gill, R. D., and Keiding, N. (1993), *Statistical Models Based on Counting Processes*, New York: Springer. 5

Braun, W. J. and Kulperger, R. J. (1997), "Properties of a Fourier bootstrap method for time series," *Communications in Statistics — Theory and Methods*, **26**, 1329–1327. 6

Canty, A. J. and Davison, A. C. (1999), "Implementation of saddlepoint approximations in resampling problems," *Statistics and Computing*, **9**, 9–15. 6

Canty, A. J., Davison, A. C., and Hinkley, D. V. (1996), "Reliable confidence intervals. Discussion of "Bootstrap confidence intervals", by T. J. DiCiccio and B. Efron," *Statistical Science*, **11**, 214–219. 4

Cox, D. R. (1972), "Regression Models and Life Tables" (with discussion), *Journal of the Royal Statistical Society series B*, **34**, 187–220. 5

Davison, A. C. (1988), "Discussion of the Royal Statistical Society meeting on the bootstrap," *Journal of the Royal Statistical Society series B*, **50**, 356–357. 3

Davison, A. C. and Hinkley, D. V. (1997), *Bootstrap Methods and Their Application*, Cambridge: Cambridge University Press. 2, 6, 7

Davison, A. C., Hinkley, D. V., and Schechtman, E. (1986), "Efficient bootstrap simulation," *Biometrika*, **73**, 555–566. 3

Efron, B. (1981), "Censored data and the bootstrap," *Journal of the American Statistical Association*, **76**, 312–319. 4

—— (1992), "Jackknife-after-bootstrap standard errors and influence functions " (with discussion), *Journal of the Royal Statistical Society series B*, **54**, 83–127. 4

Efron, B. and Tibshirani, R. J. (1993), *An Introduction to the Bootstrap*, New York: Chapman & Hall. 2

Hall, P. (1989), "On efficient bootstrap simulation," *Biometrika*, **76**, 613–617. 3

Johns, M. V. (1988), "Importance sampling for bootstrap confidence intervals," *Journal of the American Statistical Association*, **83**, 709–714. 3

Künsch, H. R. (1989), "The jackknife and bootstrap for general stationary observations," *Annals of Statistics*, **17**, 1217–1241. 5

Politis, D. N. and Romano, J. P. (1994), "The stationary bootstrap," *Journal of the American Statistical Association*, **89**, 1303–1313. 6

Angelo J. Canty
McMaster University, Hamilton, Ont, Canada
cantya@mcmaster.ca

Diagnostic Checking in Regression Relationships

by Achim Zeileis and Torsten Hothorn

Introduction

The classical linear regression model

$$y_i = x_i^\top \beta + u_i \qquad (i = 1, \dots, n) \qquad (1)$$

is still one of the most popular tools for data analysis despite (or due to) its simple structure. Although it is appropriate in many situations, there are many pitfalls that might affect the quality of conclusions drawn from fitted models or might even lead to uninterpretable results. Some of these pitfalls that are considered especially important in applied econometrics are heteroskedasticity or serial correlation of the error terms, structural changes in the regression coefficients, nonlinearities, functional misspecification or omitted variables. Therefore, a rich variety of diagnostic tests for these situations have been developed in the econometrics community, a collection of which has been implemented in the packages **lmtest** and **strucchange** covering the problems mentioned above.

These diagnostic tests are not only useful in econometrics but also in many other fields where linear regression is used, which we will demonstrate with an application from biostatistics. As Breiman (2001) argues it is important to assess the goodness-of-fit of data models, in particular not only using omnibus tests but tests designed for a certain direction of the alternative. These diagnostic checks do not have to be seen as pure significance procedures but also as an explorative tool to extract information about the structure of the data, especially in connection with residual plots or other diagnostic plots. As Brown et al. (1975) argue for the recursive CUSUM test, these procedures can "be regarded as yardsticks for the interpretation of data rather than leading to hard and fast decisions." Moreover, we will always be able to reject the null-hypothesis provided we have enough data at hand. The question is not whether the model is wrong (it always is!) but if the irregularities are serious.

The package **strucchange** implements a variety of procedures related to structural change of the regression coefficients and was already introduced in R news by Zeileis (2001) and described in more detail in Zeileis et al. (2002). Therefore, we will focus on the package **lmtest** in the following. Most of the tests and the datasets contained in the package are taken from the book of Krämer and Sonnberger (1986), which originally inspired us to write the package. Compared to the book, we implemented later versions of some tests and modern flexible interfaces for the procedures. Most of the tests are based on the OLS residuals of a linear model, which is specified by a formula argument. Instead of a formula a fitted model of class `"lm"` can also be supplied, which should work if the data are either contained in the object or still present in the workspace—however this is not encouraged. The full references for the tests can be found on the help pages of the respective function.

We present applications of the tests contained in **lmtest** to two different data sets: the first is a macroeconomic time series from the U.S. analysed by Stock and Watson (1996) and the second is data from a study on measurments of fetal mandible length discussed by Royston and Altman (1994).

U.S. macroeconomic data

Stock and Watson (1996) investigate the stability of 76 monthly macroeconomic time series from 1959 to 1993, of which we choose the department of commerce commodity price index time series `jocci` to illustrate the tests for heteroskedasticity and serial correlation. The data is treated with the same methodology as all other series considered by Stock and Watson (1996): they were transformed suitably (here by log first differences) and then an AR(6) model was fitted and analysed. The transformed series is denoted `dy` and is depicted together with a residual plot of the AR(6) model in Figure 1.

Not surprisingly, an autoregressive model is necessary as the series itself contains serial correlation, which can be shown by the Durbin-Watson test

```
R> data(jocci)
R> dwtest(dy ~ 1, data = jocci)

        Durbin-Watson test

data:  dy ~ 1
DW = 1.0581, p-value = < 2.2e-16
alternative hypothesis:
  true autocorrelation is greater than 0
```

or the Breusch-Godfrey test which also leads to a highly significant result. In the AR(6) model given by

```
R> ar6.model <-
     dy ~ dy1 + dy2 + dy3 + dy4 + dy5 + dy6
```

where the variables on the right hand side denote the lagged variables, there is no remaining serial correlation in the residuals:

```
R> bgtest(ar6.model, data = jocci)

        Breusch-Godfrey test for
        serial correlation of order 1

data:  ar6.model
LM test = 0.2, df = 1, p-value = 0.6547
```

The Durbin-Watson test is biased in dynamic models and should therefore not be applied.

The residual plot suggests that the variance of the error component increases over time, which is emphasized by all three tests for heteroskedasticity implemented in **lmtest**: the Breusch-Pagan test fits a linear regression model to the residuals and rejects if too much of the variance is explained by the auxiliary explanatory variables, which are here the squared lagged values:

```
R> var.model <-
     ~ I(dy1^2) + I(dy2^2) + I(dy3^2) +
       I(dy4^2) + I(dy5^2) + I(dy6^2)
R> bptest(ar6.model, var.model, data = jocci)

        studentized Breusch-Pagan test

data:  ar6.model
BP = 22.3771, df = 6, p-value = 0.001034
```

The Goldfeld-Quandt test `gqtest()` and the Harrison-McCabe test `hmctest()` also give highly significant p values. Whereas the Breusch-Pagan test and the Harrison-McCabe test do not assume a particular timing of the change of variance, the Goldfeld-Quandt test suffers from the same problem as the Chow test for a change of the regression

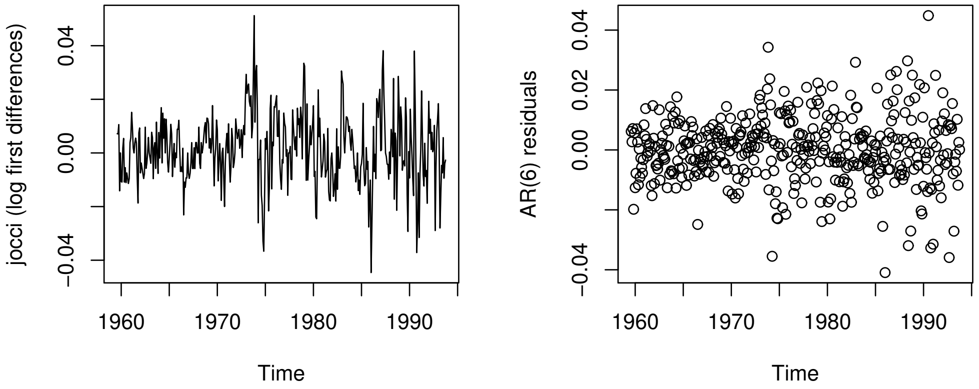

Figure 1: The jocci series and AR(6) residual plot

coefficients: the breakpoint has to be known in advance. By default it is taken to be after 50% of the observations, which leads to a significant result for the present series.

The mandible data

Royston and Altman (1994) discuss a linear regression model for data taken from a study of fetal mandible length by Chitty et al. (1993). The data comprises measurements of mandible `length` (in mm) and gestational `age` (in weeks) in 158 fetuses. The data (after log transformation) is depicted in Figure 2 together with the fitted values of a linear model `length ~ age` and a quadratic model `length ~ age + I(age^2)`.

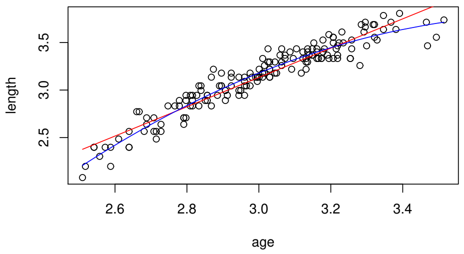

Figure 2: The mandible data

Although by merely visually inspecting the raw data or the residual plots in Figure 3 a quadratic model seems to be more appropriate, we will first fit a linear model for illustrating some tests for nonlinearity and misspecified functional form.

The suitable tests in **lmtest** are the Harvey-Collier test, which is essentially a *t* test of the recursive residuals (standardized one step prediction errors), and the Rainbow test. Both try to detect nonlinearities when the data is ordered with respect to a specific variable.

```
R> data(Mandible)
R> mandible <- log(Mandible)
R> harvtest(length ~ age, order.by = ~ age,
            data = mandible)
R> raintest(length ~ age, order.by = ~ age,
            data = mandible)
```

Both lead to highly significant results, suggesting that the model is not linear in `age`. Another appropriate procedure is the RESET test, which tests whether some auxiliary variables improve the fit significantly. By default the second and third powers of the fitted values are chosen:

```
R> reset(length ~ age, data = mandible)

        RESET test

data:  length ~ age
RESET = 26.1288, df1 = 2, df2 = 163,
  p-value = 1.436e-10
```

In our situation it would also be natural to consider powers of the regressor `age` as auxiliary variables

```
R> reset(length ~ age, power = 2,
         type = "regressor", data = mandible)

        RESET test

data:  length ~ age
RESET = 52.5486, df1 = 1, df2 = 164,
  p-value = 1.567e-11
```

which also gives a highly significant p value (higher powers do not have a significant influence). These results correspond to the better fit of the quadratic model which can both be seen in Figure 2 and 3. Although its residual plot does not look too suspicious several tests are able to reveal irregularities in this model as well. The Breusch-Pagan tests gives a p value of 0.043 and the Rainbow test gives

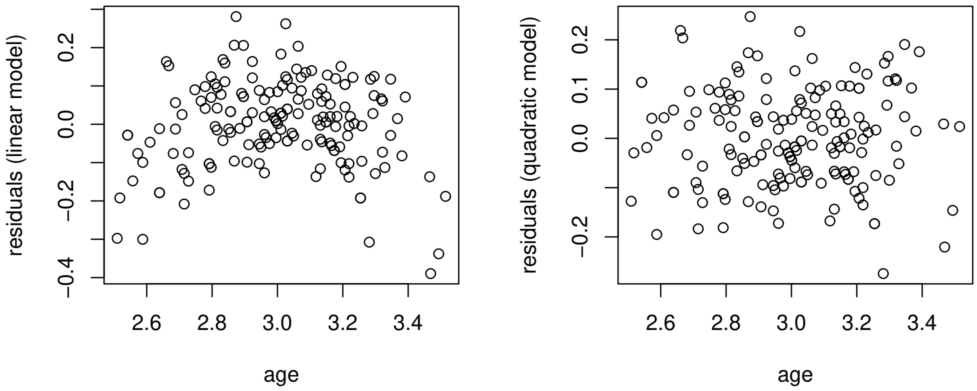

Figure 3: Residual plots for mandible models

```
R> raintest(length ~ age + I(age^2),
            order.by = ~ age, data = mandible)

        Rainbow test

data:  length ~ age + I(age^2)
Rain = 1.5818, df1 = 84, df2 = 80,
  p-value = 0.01995
```

and finally an sup$F$ test from the **strucchange** package would also reject the null hypothesis of stability at 10% level ($p = 0.064$) in favour of a breakpoint after about 90% of the observations. All three tests probably reflect that there is more variability in the edges (especially the right one) than in the middle which the model does not describe sufficiently.

Conclusions

We illustrated the usefulness of a collection of diagnostic tests for various situations of deviations from the assumptions of the classical linear regression model. We chose two fairly simple data sets—an econometric and a biometric application—to demonstrate how the tests work, but they are also particularly helpful to detect irregularities in regressions with a larger number of regressors.

Bibliography

L. Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16:199–231, 2001. 8

R. L. Brown, J. Durbin, and J. M. Evans. Techniques for testing the constancy of regression relationships over time. *Journal of the Royal Statistical Society*, B 37:149–163, 1975. 8

L. S. Chitty, S. Campbell, and D. G. Altman. Measurement of the fetal mandible – feasibility and construction of a centile chart. *Prenatal Diagnosis*, 13: 749–756, 1993. 9

W. Krämer and H. Sonnberger. *The Linear Regression Model Under Test*. Physica-Verlag, Heidelberg, 1986. 8

P. Royston and D. G. Altman. Regression using fractional polynomials of continuous covariates: Parsimonious parametric modelling. *Applied Statistics*, 43:429–453, 1994. 8, 9

J. H. Stock and M. W. Watson. Evidence on structural instability in macroeconomic time series relations. *Journal of Business & Economic Statistics*, 14:11–30, 1996. 8

A. Zeileis. `strucchange`: Testing for structural change in linear regression relationships. *R News*, 1(3):8–11, September 2001. URL `http://cran.R-project.org/doc/Rnews/`. 8

A. Zeileis, F. Leisch, K. Hornik, and C. Kleiber. `strucchange`: An R package for testing for structural change in linear regression models. *Journal of Statistical Software*, 7(2):1–38, 2002. URL `http://www.jstatsoft.org/v07/i02/`. 8

Achim Zeileis
Institut für Statistik & Wahrscheinlichkeitstheorie, Technische Universität Wien, Austria
`zeileis@ci.tuwien.ac.at`

Torsten Hothorn
Institut für Medizininformatik, Biometrie und Epidemiologie
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
`Torsten.Hothorn@rzmail.uni-erlangen.de`

Delayed Data Packages

The g.data package

by David E. Brahm

Data storage in R and S-Plus

The biggest shock for me in transitioning from S-Plus to R was the different data storage model. In S-Plus, each position in the search path corresponds to a directory ("chapter") on your disk, and every object is written immediately to disk upon creation, as a file of the same name as the object (under Unix). In R, positions on the search path are "environments", and objects live—and die—in memory unless you `save` them explicitly. The purpose of the `g.data` package is to provide data storage in R in a way that combines the best features of both models, including:

- Storing objects in individual files, without multiple `save`'s
- Viewing the contents of an attached directory without pulling them all into memory
- Loading objects into memory as they're needed, without multiple `load`'s.

For example, I store twenty large (date × stock) matrices `m1`, ..., `m20` in a "delayed data package" (DDP) called `hist`. They belong together, because they cover the same date and stock ranges, so I can calculate e.g. `mnew <- m1 * m2`. But in a given session, I may only care about two of the twenty matrices, so I don't want to load all twenty into memory. I also don't want to have to type a `load` command for every matrix I need. When I'm done, I may want to save `mnew` into the DDP for future use. And then I want to detach that DDP and attach another (with different dates and stocks), to perform the same operations there.

Here's how that looks with the `g.data` package (assuming the DDP already exists):

```
> require(g.data)
> g.data.attach("/rdata/hist")
> assign("mnew", m1*m2, 2)
> g.data.save()
> detach(2)
```

More examples

Here's an example that creates a database from scratch.

```
> g.data.attach("/tmp/newdir", warn=FALSE)
> assign("x1", matrix(1, 1000, 1000), 2)
> assign("x2", matrix(2, 1000, 1000), 2)
> g.data.save()
> detach(2)
```

In the next example, the first timed command takes a while, because `x1` is being loaded. The second is quick, because now `x1` is in memory.

```
> g.data.attach("/tmp/newdir")
> system.time(print(dim(x1)))
[1] 1000 1000
[1] 1.70 0.01 1.90 0.00 0.00
> system.time(print(dim(x1)))
[1] 1000 1000
[1] 0 0 0 0 0
```

Suppose you now type

```
> assign("x3", x2*10, 2)
```

and go look at the contents of '/tmp/newdir'. Objects `x1` and `x2` are written there (under subdirectory 'data', with names 'x1.RData' and 'x2.RData'), but `x3` is not. 'x3.RData' only gets written when you type `g.data.save()`. Unlike S-Plus, you have the option *not* to save the changes you've made, just by not calling `g.data.save`.

With no arguments, `g.data.save` (re-)writes all objects in position 2 to disk. Both `g.data.attach` and `g.data.save` take an optional argument `pos` (defaulting to `pos=2`), so you can work with multiple DDP's in different positions on the search path. `g.data.save` also takes an argument `obj` to (re-)write only specified objects, an argument `rm.obj` to remove objects, and an argument `dir` which can be used e.g. in this context:

```
> y <- list(a=1, b=11:15, c=21:29)
> attach(y, pos=4)
> g.data.save(dir="/tmp/mylist", pos=4)
```

Finally, the command `g.data.get` allows you to retrieve individual objects stored in a DDP without attaching the DDP:

```
> bcopy <- g.data.get("b", "/tmp/mylist")
> bcopy
[1] 11 12 13 14 15
```

Under the hood

`g.data.save` creates (or writes to) a DDP directory with subdirectories 'R' and 'data', so it looks to R much like a typical package. `g.data.attach` is really just a snippet of `library`. The data files in subdirectory 'data' are created with `save`, as you'd expect. The code under subdirectory 'R' consists of lines like:

```
x1 <- delay(g.data.load("x1", "newdir"))
```

so when you first `g.data.attach` the directory, the object loaded into memory as `x1` is a "promise object" (which is very small). When you actually use `x1` (e.g., to query its dimensions), the promise is fulfilled, and `g.data.load` does two things:

1. It loads the actual large object, and
2. It returns that object for the query.

Henceforth, the object in memory as `x1` is the real (large) object.

David E. Brahm
Geode Capital Management
brahm@alum.mit.edu

geepack: Yet Another Package for Generalized Estimating Equations

Modeling Both Mean and Association of Multivariate Responses

by Jun Yan

Introduction

geepack is designed to provide an inferential basis for both the association structure and the mean structure in multivariate analysis, using the Generalized Estimating Equations (GEE) approach.

Consider a sample of K independent clusters $y_i^T = (y_{i1}, \cdots, y_{in_i})$, $i = 1, \cdots, K$, of n_i-variate responses. In a generalized linear model setup, the variance of y_{it}, V_{it}, can be factored as

$$\mathrm{var}(y_{it}) = \phi_{it} v(\mu_{it}),$$

where ϕ_{it} is the scale parameter, v is the variance function $v(\mu_{it})$, where $\mu_{it} = E(y_{it})$. To model the association, we decompose $\mathrm{cov}(y_i)$ into two parts, the variance and the correlation,

$$\mathrm{cov}(y_i) = V^{1/2} R V^{1/2},$$

where V is the diagonal matrix of V_{it}, and R is the correlation matrix of y_i.

Let X_{1i}, X_{2i} and X_{3i} be the covariate matrices for the mean, the scale, and the correlation of the response y_i, with dimensions $n_i \times p$, $n_i \times r$, and $n_i(n_i - 1)/2 \times q$, respectively. The models are

$$g_1(\mu_i) = X_{1i}\beta, \tag{1}$$
$$g_2(\phi_i) = X_{2i}\gamma, \tag{2}$$
$$g_3(\rho_i) = X_{3i}\alpha, \tag{3}$$

where $g_i, i = 1, 2, 3$, are known link functions, μ_i is a $n_i \times 1$ vector containing $E(y_i|X_{1i})$, ϕ_i is a $n_i \times 1$ vector containing $\mathrm{var}(y_i|X_{2i})/v_{it}$, where $v_{it} = v(\mu_{it})$ is the variance function, and ρ_i is a $n_i(n_i - 1)/2 \times 1$ vector containing $\mathrm{cor}(y_{is}, y_{it}|X_{3i})$. β, γ, and α are the mean, the scale, and the correlation parameters of dimension $p \times 1$, $r \times 1$, and $q \times 1$, respectively.

The mean link has been well studied. The scale link is often taken to be log, while it is natural to let the correlation link be "logistic" (i.e., Fisher's z transformation), in which case the inverse link function is the hyperbolic tangent, that is,

$$\rho_{its} = \mathrm{cor}(y_{is}, y_{it}|X_{3i}) = \frac{\exp(X_{3i(s,t)}\alpha) - 1}{\exp(X_{3i(s,t)}\alpha) + 1}, \tag{4}$$

where $X_{3i(s,t)}$ is the row in matrix X_{3i} corresponding to the correlation of y_{is} and y_{it}. These links ensure that the scale is positive and that the correlation is in $(-1, 1)$. The scale model is useful in situations where parameters are needed for covariate effects either on over- or under-dispersion or on heteroscedasticity.

A convenient set of estimating equations for the three-link model is

$$U_1(\beta, \gamma, \alpha) = \sum_{i=1}^{K} D_{1i}^T V_{1i}^{-1}(y_i - \mu_i) = 0 \tag{5}$$
$$U_2(\beta, \gamma, \alpha) = \sum_{i=1}^{K} D_{2i}^T V_{2i}^{-1}(s_i - \phi_i) = 0 \tag{6}$$
$$U_3(\beta, \gamma, \alpha) = \sum_{i=1}^{K} D_{3i}^T V_{3i}^{-1}(z_i - \rho_i) = 0 \tag{7}$$

where s_i is the $n_i \times 1$ vector of $s_{it} = (y_{it} - \mu_{it})^2/v_{it}$, z_i is the $n_i(n_i - 1)/2 \times 1$ vector of $z_{its} = (y_{it} - \mu_{it})(y_{is} - \mu_{is})/\sqrt{\phi_{it}v_{it}\phi_{is}v_{is}}$, $D_{1i} = \partial\mu_i/\partial\beta^T$, $D_{2i} = \partial\phi_i/\partial\gamma^T$, $D_{3i} = \partial\rho_i/\partial\alpha^T$, and V_{1i}, V_{2i} and V_{3i} are the conditional working covariance matrices of y_i, s_i, and z_i.

The matrix V_{1i} generally contains scale parameters γ and correlation parameters α. The matrices V_{2i} and V_{3i} may contain other estimated quantities which characterize the third and fourth order moments. In practice, in order to avoid specification of higher order moments, estimation of higher order nuisance parameters, and convergence problems, V_{2i} may be chosen to be a diagonal matrix whose diagonal elements are $2\phi_{it}$, following the independence Gaussian working matrix in Prentice and Zhao (1991), and V_{3i} may be an identity matrix (Ziegler et al., 1998, p.129), at the cost of potential efficiency loss. These simplifications are implemented in **geepack**.

Features

- Allows different covariates in separate models for the mean, scale, and correlation via various link functions.
- Provides "sandwich" and jackknife variance estimators for all the parameter estimates, extending Ziegler et al. (2000).
- Handles clustered ordinal data, allowing covariates in the odds ratio model, using the method in Heagerty and Zeger (1996).

An example: Epileptic seizures

As an illustration, the epileptic seizure data (Thall and Vail, 1990) is analyzed. The dataset arose from a clinical trial of 59 epileptic patients, randomized to receive either the anti-epileptic drug pragabide or a placebo, as an adjuvant to standard chemotherapy. There are four 2-week interval seizure counts for each patient. The covariates are treatment, age, and baseline counts on a 8-week interval before the trial. We first reshape the data into a "long" format for longitudinal data, and create new covariates identical to those used in Thall and Vail (1990),

```
> data(seizure)
> seiz.l <-
+   reshape(seizure,
+           varying = list(c("y1", "y2",
+                            "y3", "y4")),
+           v.names = "y", direction = "long")
> seiz.l <-
+   seiz.l[order(seiz.l$id, seiz.l$time),]
> seiz.l$lbase <- log(seiz.l$base / 4)
> seiz.l$lage <- log(seiz.l$age)
> seiz.l$v4 <- ifelse(seiz.l$time == 4, 1, 0)
```

Next we use the function geese to fit a GEE model for the seizure counts, with the same mean model as that in Thall and Vail (1990). We treat time as a factor and include it in the scale model using a `log` link. To illustrate the usage of the correlation model, we use an `ar1` correlation structure (together with the scale model, this specifies a heterogeneous AR(1) covariance structure), and `fisherz` link, and include patient age in the correlation model. The models for the mean, scale, and correlation are fit using GEE and the results are summarized in the following.

```
> z <- model.matrix(~ age, data = seizure)
> m1 <- geese(y ~ lbase*trt + lage + v4,
+             sformula = ~ as.factor(time) - 1,
+             id = id, data = seiz.l,
+             corstr = "ar1", family = poisson,
+             zcor = z, cor.link = "fisherz",
+             sca.link = "log")
> summary(m1)

Mean Model:
 Mean Link:                 log
 Variance to Mean Relation: poisson

 Coefficients:
            estimate san.se   wald        p
(Intercept)   -2.544 0.8291   9.41 0.002154
lbase          0.964 0.0898 115.22 0.000000
trt           -1.491 0.4557  10.71 0.001068
lage           0.826 0.2387  11.98 0.000539
v4            -0.143 0.0721   3.95 0.046850
lbase:trt      0.601 0.1864  10.38 0.001272

Scale Model:
 Scale Link:                log

 Estimated Scale Parameters:
                 estimate san.se wald        p
as.factor(time)1    1.240  0.255 23.6 1.18e-06
as.factor(time)2    1.544  0.366 17.8 2.49e-05
as.factor(time)3    2.019  0.498 16.5 4.98e-05
as.factor(time)4    0.864  0.204 18.0 2.24e-05

Correlation Model:
 Correlation Structure:     ar1
 Correlation Link:          fisherz

 Estimated Correlation Parameters:
            estimate san.se  wald        p
(Intercept)    2.558 0.7064 13.11 0.000294
age           -0.047 0.0229  4.21 0.040192

Returned Error Value:    0
Number of clusters: 59  Maximum cluster size: 4
```

Since there is one possible outlier in the dataset (Diggle et al., 1994, pp.166–168), it might be interesting to compare the "sandwich" variance estimate with the jackknife variance estimate. Jackknife variance estimate may be obtained by setting `jack`, `j1s`, or `fij` to `TRUE`, requesting approximated, one-step, and fully iterated jackknife variance estimate, respectively; see Ziegler et al. (2000).

```
> m2 <- geese(y ~ lbase*trt + lage + v4,
+             sformula = ~ as.factor(time) - 1,
+             id = id, data = seiz.l,
+             corstr = "ar1", family = poisson,
+             zcor = z, cor.link = "fisherz",
+             sca.link = "log", jack = TRUE,
+             j1s = TRUE, fij = TRUE)
```

Summarizing the fitted object (not shown here) suggests that there is noticeable difference between the sandwich and the jackknife variance estimator for the covariate effect of `trt` and `lbase:trt`. If the jackknife variance estimators were used, these two effects would become insignificant at level 0.05.

Future developments

Different components within a cluster may have different link functions. For example, the data analyzed by Prentice and Zhao (1991) have two responses for each patient. One is continuous and its mean is modeled with the identity link, and the other is binary and its mean is modeled with the logit link. The C++ code for **geepack** was designed to permit this situation. An R interface will be developed for this extension.

Acknowledgments

The C++ code in **geepack** is based on the Template Numerical Toolkit (TNT) version 0.94, developed at the National Institute of Standards and Technology (NIST). At the time of writing, version 1.1 is available at `http://math.nist.gov/tnt/`.

I am grateful to Douglas Bates and Jason Fine for encouragement, discussions, and comments.

Bibliography

Peter J. Diggle, Kung-Yee Liang, and Scott L. Zeger. *Analysis of longitudinal data (ISBN 0198522843)*. Clarendon Press [Oxford University Press], 1994. ISBN 0198522843. 13

Patrick J. Heagerty and Scott L. Zeger. Marginal regression models for clustered ordinal measurements. *Journal of the American Statistical Association*, 91:1024–1036, 1996. 13

Ross L. Prentice and Lue Ping Zhao. Estimating equations for parameters in means and covariances of multivariate discrete and continuous responses. *Biometrics*, 47:825–839, 1991. 12, 14

Peter F. Thall and Stephen C. Vail. Some covariance models for longitudinal count data with overdispersion. *Biometrics*, 46:657–671, 1990. 13

Andreas Ziegler, Christian Kastner, and Maria Blettner. The generalised estimating equations: An annotated bibliography. *Biometrical Journal*, 40:115–139, 1998. 12

Andreas Ziegler, Christian Kastner, Daniel Brunner, and Maria Blettner. Familial associations of lipid profiles: A generalized estimating equations approach. *Statistics in Medicine*, 19(24):3345–3357, 2000. 13

Jun Yan
University of Wisconsin–Madison, U.S.A.
`jyan@stat.wisc.edu`

On Multiple Comparisons in R

by Frank Bretz, Torsten Hothorn and Peter Westfall

Description

The multiplicity problem arises when several inferences are considered simultaneously as a group. If each inference has a 5% error rate, then the error rate over the entire group can be much higher than 5%. This article shows practical examples of multiple comparisons procedures that control the error of making any incorrect inference.

The **multcomp** package for the R statistical environment allows for multiple comparisons of parameters whose estimates are generally correlated, including comparisons of k groups in general linear models. The package has many common multiple comparison procedures "hard-coded", including Dunnett, Tukey, sequential pairwise contrasts, comparisons with the average, changepoint analysis, Williams', Marcus', McDermott's, and tetrad contrasts. In addition, a free input interface for the contrast matrix allows for more general comparisons.

The comparisons themselves are not restricted to balanced or simple designs. Instead, the package is designed to provide general multiple comparisons, thus allowing for covariates, nested effects, correlated means, likelihood-based estimates, and missing values. For the homoscedastic normal linear models, the functions in the package account for the correlations between test statistics by using the exact multivariate t-distribution. The resulting procedures are therefore more powerful than the Bonferroni and Holm methods; adjusted p-values for these methods are reported for reference. For more general models, the program accounts for correlations using the asymptotic multivariate normal distribution; examples include multiple comparisons based on rank transformations, logistic regression, GEEs, and proportional hazards models. In the asymptotic case, the user must supply the estimates, the asymptotic covariance matrix, and the contrast matrix.

Basically, the package provides two functions. The first, `simint`, computes confidence intervals for the common single-step procedures. This approach is uniformly improved by the second function (`simtest`), which utilizes logical constraints and is closely related to closed testing. However, no confidence intervals are available for the `simtest` function. For testing and validation purposes, some examples from Westfall et al. (1999) are included in the package.

Details

Assume the general linear model

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where $\mathbf{Y}$ is the $n \times 1$ observation vector, $\mathbf{X}$ is the fixed and known $n \times p$ design matrix, $\boldsymbol{\beta}$ is the fixed and unknown $p \times 1$ parameter vector and $\boldsymbol{\epsilon}$ is the random, unobservable $n \times 1$ error vector, distributed as $N_n(\mathbf{0}, \sigma^2\mathbf{I}_n)$. We assume the usual estimates

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^t\mathbf{X})^- \mathbf{X}^t\mathbf{Y}$$

and

$$\hat{\sigma}^2 = (\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}})^t(\mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}})/\nu,$$

where $\nu = n - \mathrm{rank}(\mathbf{X})$. Our focus is on multiple comparisons for parameters of the general form $\mathbf{c}^t\boldsymbol{\beta}$. Its variance is given through

$$\mathrm{Var}(\mathbf{c}^t\hat{\boldsymbol{\beta}}) = \hat{\sigma}^2\mathbf{c}^t(\mathbf{X}^t\mathbf{X})^-\mathbf{c}.$$

In simultaneous inferences we are faced with a given family of estimable parameters $\{\mathbf{c}_1^t\boldsymbol{\beta}, \ldots, \mathbf{c}_k^t\boldsymbol{\beta}\}$. We thus use the pivotal test statistics

$$T_i = \frac{\mathbf{c}_i^t\hat{\boldsymbol{\beta}} - \mathbf{c}_i^t\boldsymbol{\beta}}{\hat{\sigma}\sqrt{\mathbf{c}_i^t(\mathbf{X}^t\mathbf{X})^-\mathbf{c}_i}}.$$

For a general account on multiple comparison procedures we refer to Hochberg and Tamhane (1987). The joint distribution of $\{T_1, \ldots, T_k\}$ is multivariate t with degrees of freedom ν and correlation matrix $\mathbf{R} = \mathbf{D}\mathbf{C}(\mathbf{X}^t\mathbf{X})^-\mathbf{C}^t\mathbf{D}$, where $\mathbf{C}^t = (c_1, \ldots, c_k)$ and $\mathbf{D} = \mathrm{diag}(\mathbf{c}_i^t(\mathbf{X}^t\mathbf{X})^-\mathbf{c}_i)^{-1/2}$. In the asymptotic case $\nu \to \infty$ or if σ is known, the corresponding limiting multivariate normal distribution holds. The numerical evaluation of the multivariate t and normal distribution is available with the R package **mvtnorm**, see Hothorn et al. (2001).

The function `simint` provides simultaneous confidence intervals for the estimable functions $\mathbf{c}_i^t\boldsymbol{\beta}$ in the (two-sided) form

$$[\mathbf{c}_i^t\hat{\boldsymbol{\beta}} - c_{1-\alpha}\hat{\sigma}\sqrt{\mathbf{c}_i^t(\mathbf{X}^t\mathbf{X})^-\mathbf{c}_i}; \quad \mathbf{c}_i^t\hat{\boldsymbol{\beta}} + c_{1-\alpha}\hat{\sigma}\sqrt{\mathbf{c}_i^t(\mathbf{X}^t\mathbf{X})^-\mathbf{c}_i}],$$

where $c_{1-\alpha}$ is the critical value at level $1 - \alpha$, as derived under the distributional assumptions above. If lower or upper tailed tests are used, the corresponding interval bounds are set to $-\infty$ and ∞, respectively.

The second function `simtest` provides more powerful test decisions than `simint` yet it does not provide simultaneous confidence intervals. It uses the stepwise methods of Westfall (1997), which take the logical constraints between the hypotheses into account and which are closely related to the closed testing principle of Marcus et al. (1976). In addition, the stochastic dependencies of the test statistics are incorporated, thus allowing imbalance, covariates and more general models. Again, any collection of linear combinations of the estimable parameters is allowed, not just pairwise comparisons. We refer to Westfall (1997) for the algebraic and algorithmic details.

Example

We illustrate some of the capabilities of the **multcomp** package using the `recovery` dataset. Three different heating blankets b_1, b_2, b_3 for post-surgery treatment are compared to a standard blanket b_0. The variable of interest in this simple one-way layout was recovery time in minutes of patients allocated randomly to one of the four treatments. The standard approach for comparing several treatments against a control is the many-to-one test of Dunnett (1955). The Dunnett test is one of the "hard-coded" procedures available for one-factor models in **multcomp**. To obtain simultaneous confidence intervals for the comparisons $\beta_i - \beta_1$ on simply calls:

```
>library(multcomp)

Loading required package: mvtnorm

>data(recovery)
>Dcirec <- simint(minutes ~ blanket,
+      data = recovery, conf.level = 0.9,
+      alternative = "less")

>print(Dcirec)

        Simultaneous confidence
        intervals: Dunnett contrasts

        90 % confidence intervals

                    Estimate lower CI
blanketb1-blanketb0   -2.133     -Inf
blanketb2-blanketb0   -7.467     -Inf
blanketb3-blanketb0   -1.667     -Inf
                    upper CI
blanketb1-blanketb0    0.822
blanketb2-blanketb0   -4.511
blanketb3-blanketb0   -0.036
```

Thus, blankets b_2 and b_3 lead to significant lower recovery times in comparison to the standard b_0, since the respective upper confidence bounds are less than 0. In particular, the output above indicates that at the designated confidence level of 90% the average recovery time for b_2 is more than 7 minutes shorter than it is for b_0.

A second way to obtain the same results is to define the contrast matrix **C** explicitly:

```
>C <- matrix(c(0, 0, 0, -1, -1,
+     -1, 1, 0, 0, 0, 1, 0, 0, 0,
+     1), nc = 5)
>rownames(C) <- paste("C", 1:nrow(C),
+     sep = "")
>Ccirec <- simint(minutes ~ blanket,
+     data = recovery, conf.level = 0.9,
+     alternative = "less", eps = 1e-04,
+     cmatrix = C)

>print(Ccirec)

        Simultaneous confidence
        intervals: user-defined contrasts

        90 % confidence intervals

   Estimate lower CI upper CI
C1  -2.1333     -Inf   0.8225
C2  -7.4667     -Inf  -4.5108
C3  -1.6667     -Inf  -0.0360
```

The first column of **C** stands for the intercept β_0, the remaining columns are reserved for the 4 levels $\beta_1, \ldots, \beta_4$ of the single factor. Each row defines a particular linear combination $\mathbf{c}_i^t\boldsymbol{\beta}$. Note that the *eps* argument specifies the accuracy of the numerical results (see `pmvt` in package `mvtnorm` for more details). This is the reason why the confidence bounds are now printed with four significant digits instead of the former three digits.

More detailed output is available by using the summary method:

```
>summary(Ccirec)

        Simultaneous 90% confidence
        intervals: user-defined contrasts

        user-defined contrasts for factor blanket

Contrast matrix:
   [,1] [,2] [,3] [,4] [,5]
C1    0   -1    1    0    0
C2    0   -1    0    1    0
C3    0   -1    0    0    1

Absolute Error Tolerance:  1e-04

 90 % quantile:  1.8431

Coefficients:
   Estimate low CI,  upp CI t value
C1  -2.1333    -Inf  0.8225 -1.3302
C2  -7.4667    -Inf -4.5108 -4.6556
C3  -1.6667    -Inf -0.0360 -1.8837
   Std.Err.  p raw p Bonf  p adj
C1   1.6038 0.0958 0.2874 0.2412
C2   1.6038 0.0000 0.0001 0.0001
C3   0.8848 0.0337 0.1012 0.0924
```

This output prints the user defined contrast matrix **C** and the quantile $c_{1-\alpha}$. In addition, simultaneous confidence intervals, the estimates $\mathbf{c}_i^t\hat{\boldsymbol{\beta}}$ and their standard errors are given as well as the raw p-values (computed from the marginal t distributions) and multiplicity adjusted p-values (using either the multivariate t distribution or the Bonferroni correction). The simultaneous confidence intervals and the adjusted p-values based on the multivariate t distribution are compatible in the sense that if $p_{adj} < 0.05$, then the associated confidence interval does not contain the 0.

A more powerful approach is available using the `simtest` function. The call remains essentially the same, also no simultaneous confidence intervals are available:

```
>Ctrec <- simtest(minutes ~ blanket,
+     data = recovery, conf.level = 0.9,
+     alternative = "less", eps = 1e-04,
+     cmatrix = C)

>summary(Ctrec)

        Simultaneous tests: user-defined contrasts

        user-defined contrasts for factor blanket

Contrast matrix:
   [,1] [,2] [,3] [,4] [,5]
C1    0   -1    1    0    0
C2    0   -1    0    1    0
C3    0   -1    0    0    1


Absolute Error Tolerance:  1e-04

Coefficients:
   Estimate t value Std.Err.  p raw
C2  -7.4667 -4.6556   1.6038 0.0000
C3  -1.6667 -1.8837   1.6038 0.0337
C1  -2.1333 -1.3302   0.8848 0.0958
   p Bonf  p adj
C2 0.0001 0.0001
C3 0.0675 0.0640
C1 0.0958 0.0958
```

It transpires that the adjusted p-values are indeed uniformly lower in comparison to those from `simint`.

A final example call illustrates the use of the **multcomp** package, if the estimates $\hat{\beta}_i$ and their covariances are passed by hand. In such cases, the core functions `csimint` and `csimtest` have to be called without using the `sim{int,test}` interfaces. The call

```
>parm <- c(14.8, 12.6667, 7.3333,
+     13.1333)
>N <- c(20, 3, 3, 15)
>contrast <- contrMat(N, type = "Dunnett")
>nu <- 37
>mse <- 6.7099
```

```
>covm <- mse * diag(1/N)
>csimint(estpar = parm, df = as.integer(nu),
+      covm = covm, cmatrix = contrast,
+      conf.level = 0.9, alternative = "less")

        Simultaneous confidence
        intervals: user-defined contrasts

        90 % confidence intervals

    Estimate lower CI upper CI
2-1   -2.133     -Inf     0.823
3-1   -7.467     -Inf    -4.511
4-1   -1.667     -Inf    -0.036
```

yields the same result as the first call above. The sample size vector N and the mean square error *mse* are only required for a convenient computation of the covariance matrix. Note that the contrast matrix can either be entered by hand or by using the availability of standard contrast matrices in the `contrMat` function.

Graphical Representation

The method `plot.hmtest` is available for a graphical inspectation of the simultaneous confidence intervals. For each contrast, the confidence interval is plotted, for example `plot(Dcirec)` can be used for plotting the one-sided Dunnett confidence intervals for the `recovery` example from the first code snippet.

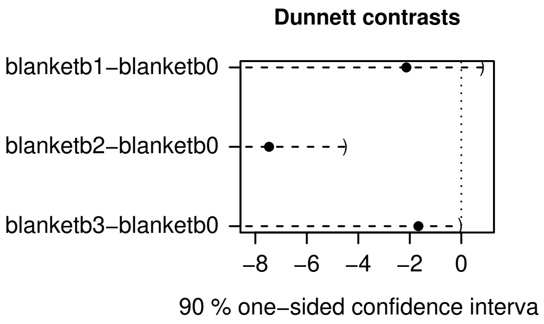

Figure 1: A graphical representation of one-sided Dunnett confidence intervals. The intervals are plotted as horizontal lines where the limits of the intervals are given by round brackets and the estimates by a point.

Conclusion

This article addressed the application of multiple comparisons using the **multcomp** package. The present methods cover several standard test procedures and allow for user specified type of comparisons. Also the discussion has been devoted to general linear models, the package is also applicable to more general linear and nonlinear mixed models as long as the covariances between the estimates are known.

Currently, the quantiles of the multivariate t or normal distribution are computed using `uniroot` on the p-value functions. This is time consuming and will be improved in future versions of the **mvtnorm** package.

Bibliography

Y. Hochberg and A. Tamhane. *Multiple comparison procedures*. Wiley, New York, 1987. 15

T. Hothorn, F. Bretz, and A. Genz. On multivariate t and Gauss probabilities in R. *R News*, 1(2):27–29, 2001. 15

R. Marcus, E. Peritz, and K.B. Gabriel. On closed testing procedures with special reference to ordered analysis of variance. *Biometrika*, 63:655–660, 1976. 15

P. Westfall. Multiple testing of general contrasts using logical constraints and correlations. *Journal of the American Statistical Association*, 92:299–306, 1997. 15

P. H. Westfall, R. D. Tobias, D. Rom, R. D. Wolfinger, and Y. Hochberg. *Multiple Comparisons and Multiple Tests Using the SAS System*. SAS Institute Inc., Cary, NC, 1999. 15

Frank Bretz
Universität Hannover, LG Bioinformatik, FB Gartenbau
Herrenhäuser Str. 2, D-30419 Hannover
`bretz@ifgb.uni-hannover.de`

Torsten Hothorn
Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Medizininformatik, Biometrie und Epidemiologie, Waldstraße 6, D-91054 Erlangen
`Torsten.Hothorn@rzmail.uni-erlangen.de`

Peter Westfall
Texas Tech University, Department of Information Systems and Quantitative Sciences, Lubbock, TX 79409
`WESTFALL@ba.ttu.edu`

We would like to thank Doug Bates for corrections and suggestions improving the readability.

Classification and Regression by randomForest

Andy Liaw and Matthew Wiener

Introduction

Recently there has been a lot of interest in "ensemble learning" — methods that generate many classifiers and aggregate their results. Two well-known methods are boosting (see, e.g., Shapire et al., 1998) and bagging Breiman (1996) of classification trees. In boosting, successive trees give extra weight to points incorrectly predicted by earlier predictors. In the end, a weighted vote is taken for prediction. In bagging, successive trees do not depend on earlier trees — each is independently constructed using a bootstrap sample of the data set. In the end, a simple majority vote is taken for prediction.

Breiman (2001) proposed random forests, which add an additional layer of randomness to bagging. In addition to constructing each tree using a different bootstrap sample of the data, random forests change how the classification or regression trees are constructed. In standard trees, each node is split using the best split among all variables. In a random forest, each node is split using the best among a subset of predictors randomly chosen at that node. This somewhat counterintuitive strategy turns out to perform very well compared to many other classifiers, including discriminant analysis, support vector machines and neural networks, and is robust against overfitting (Breiman, 2001). In addition, it is very user-friendly in the sense that it has only two parameters (the number of variables in the random subset at each node and the number of trees in the forest), and is usually not very sensitive to their values.

The **randomForest** package provides an R interface to the Fortran programs by Breiman and Cutler (available at `http://www.stat.berkeley.edu/users/breiman/`). This article provides a brief introduction to the usage and features of the R functions.

The algorithm

The random forests algorithm (for both classification and regression) is as follows:

1. Draw $n_{\texttt{tree}}$ bootstrap samples from the original data.
2. For each of the bootstrap samples, grow an *unpruned* classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample $m_{\texttt{try}}$ of the predictors and choose the best split from among those variables. (Bagging can be thought of as the special case of random forests obtained when $m_{\texttt{try}} = p$, the number of predictors.)
3. Predict new data by aggregating the predictions of the $n_{\texttt{tree}}$ trees (i.e., majority votes for classification, average for regression).

An estimate of the error rate can be obtained, based on the training data, by the following:

1. At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls "out-of-bag", or OOB, data) using the tree grown with the bootstrap sample.
2. Aggregate the OOB predictions. (On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions.) Calcuate the error rate, and call it the OOB estimate of error rate.

Our experience has been that the OOB estimate of error rate is quite accurate, given that enough trees have been grown (otherwise the OOB estimate can bias upward; see Bylander (2002)).

Extra information from Random Forests

The **randomForest** package optionally produces two additional pieces of information: a measure of the importance of the predictor variables, and a measure of the internal structure of the data (the proximity of different data points to one another).

Variable importance This is a difficult concept to define in general, because the importance of a variable may be due to its (possibly complex) interaction with other variables. The random forest algorithm estimates the importance of a variable by looking at how much prediction error increases when (OOB) data for that variable is permuted while all others are left unchanged. The necessary calculations are carried out tree by tree as the random forest is constructed. (There are actually four different measures of variable importance implemented in the classification code. The reader is referred to Breiman (2002) for their definitions.)

proximity measure The (i, j) element of the proximity matrix produced by **randomForest** is the fraction of trees in which elements i and j fall in the same terminal node. The intuition is that "similar" observations should be in the same terminal nodes more often than dissimilar ones. The proximity matrix can be used to identify structure in the data (see Breiman, 2002) or for unsupervised learning with random forests (see below).

Usage in R

The user interface to random forest is consistent with that of other classification functions such as `nnet()` (in the **nnet** package) and `svm()` (in the **e1071** package). (We actually borrowed some of the interface code from those two functions.) There is a formula interface, and predictors can be specified as a matrix or data frame via the `x` argument, with responses as a vector via the `y` argument. If the response is a factor, `randomForest` performs classification; if the response is continuous (that is, not a factor), `randomForest` performs regression. If the response is unspecified, `randomForest` performs unsupervised learning (see below). Currently `randomForest` does not handle ordinal categorical responses. Note that categorical predictor variables must also be specified as factors (or else they will be wrongly treated as continuous).

The `randomForest` function returns an object of class `"randomForest"`. Details on the components of such an object are provided in the online documentation. Methods provided for the class includes `predict` and `print`.

A classification example

The Forensic Glass data set was used in Chapter 12 of MASS4 (Venables and Ripley, 2002) to illustrate various classification algorithms. We use it here to show how random forests work:

```
> library(randomForest)
> library(MASS)
> data(fgl)
> set.seed(17)
> fgl.rf <- randomForest(type ~ ., data = fgl,
+     mtry = 2, importance = TRUE,
+     do.trace = 100)
 100: OOB error rate=20.56%
 200: OOB error rate=21.03%
 300: OOB error rate=19.63%
 400: OOB error rate=19.63%
 500: OOB error rate=19.16%
> print(fgl.rf)
Call:
 randomForest.formula(formula = type ~ .,
   data = fgl, mtry = 2, importance = TRUE,
   do.trace = 100)
           Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 19.16%
Confusion matrix:
      WinF WinNF Veh Con Tabl Head class.error
WinF    63     6   1   0    0    0   0.1000000
WinNF    9    62   1   2    2    0   0.1842105
Veh      7     4   6   0    0    0   0.6470588
Con      0     2   0  10    0    1   0.2307692
Tabl     0     2   0   0    7    0   0.2222222
Head     1     2   0   1    0   25   0.1379310
```

We can compare random forests with support vector machines by doing ten repetitions of 10-fold cross-validation, using the `errorest` functions in the **ipred** package:

```
> library(ipred)
> set.seed(131)
> error.RF <- numeric(10)
> for(i in 1:10) error.RF[i] <-
+     errorest(type ~ ., data = fgl,
+     model = randomForest, mtry = 2)$error
> summary(error.RF)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1869  0.1974  0.2009  0.2009  0.2044  0.2103
> library(e1071)
> set.seed(563)
> error.SVM <- numeric(10)
> for (i in 1:10) error.SVM[i] <-
+     errorest(type ~ ., data = fgl,
+     model = svm, cost = 10, gamma = 1.5)$error
> summary(error.SVM)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2430  0.2453  0.2523  0.2561  0.2664  0.2710
```

We see that the random forest compares quite favorably with SVM.

We have found that the variable importance measures produced by random forests can sometimes be useful for model reduction (e.g., use the "important" variables to build simpler, more readily interpretable models). Figure 1 shows the variable importance of the Forensic Glass data set, based on the `fgl.rf` object created above. Roughly, it is created by

```
> par(mfrow = c(2, 2))
> for (i in 1:4)
+   plot(sort(fgl.rf$importance[,i], dec = TRUE),
+   type = "h", main = paste("Measure", i))
```

We can see that measure 1 most clearly differentiates the variables. If we run random forest again dropping `Na`, `K`, and `Fe` from the model, the error rate remains below 20%.

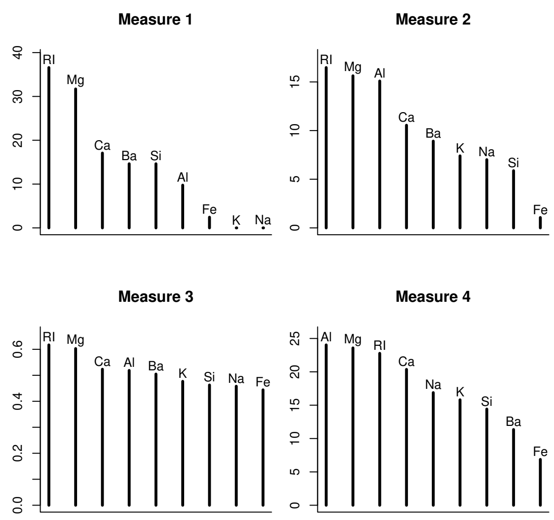

Figure 1: Variable importance for the Forensic Glass data.

The gain can be far more dramatic when there are more predictors. In a data set with thousands of predictors, we used the variable importance measures to select only dozens of predictors, and we were able to retain essentially the same prediction accuracy. For a simulated data set with 1,000 variables that we constructed, random forest, with the default $m_{\mathtt{try}}$, we were able to clearly identify the only two informative variables and totally ignore the other 998 noise variables.

A regression example

We use the Boston Housing data (available in the **MASS** package) as an example for regression by random forest. Note a few differences between classification and regression random forests:

- The default $m_{\mathtt{try}}$ is $p/3$, as opposed to $p^{1/2}$ for classification, where p is the number of predictors.
- The default `nodesize` is 5, as opposed to 1 for classification. (In the tree building algorithm, nodes with fewer than `nodesize` observations are not splitted.)
- There is only one measure of variable importance, instead of four.

```
> data(Boston)
> set.seed(1341)
> BH.rf <- randomForest(medv ~ ., Boston)
> print(BH.rf)
Call:
 randomForest.formula(formula = medv ~ .,
     data = Boston)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 10.64615
                    % Var explained: 87.39
```

The "mean of squared residuals" is computed as

$$\mathrm{MSE}_{\mathrm{OOB}} = n^{-1} \sum_{1}^{n} \{y_i - \hat{y}_i^{\mathrm{OOB}}\}^2,$$

where $\hat{y}_i^{\mathrm{OOB}}$ is the average of the OOB predictions for the ith observation. The "percent variance explained" is computed as

$$1 - \frac{\mathrm{MSE}_{\mathrm{OOB}}}{\hat{\sigma}_y^2},$$

where $\hat{\sigma}_y^2$ is computed with n as divisor (rather than $n-1$).

We can compare the result with the actual data, as well as fitted values from a linear model, shown in Figure 2.

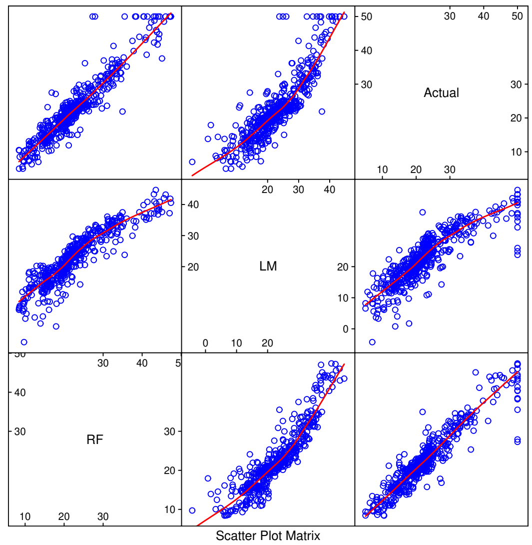

Figure 2: Comparison of the predictions from random forest and a linear model with the actual response of the Boston Housing data.

An unsupervised learning example

Because random forests are collections of classification or regression trees, it is not immediately apparent how they can be used for unsupervised learning. The "trick" is to call the data "class 1" and construct a "class 2" synthetic data, then try to classify the combined data with a random forest. There are two ways to simulate the "class 2" data:

1. The "class 2" data are sampled from the product of the marginal distributions of the variables (by independent bootstrap of each variable separately).

2. The "class 2" data are sampled uniformly from the hypercube containing the data (by sampling uniformly within the range of each variables).

The idea is that real data points that are similar to one another will frequently end up in the same terminal node of a tree — exactly what is measured by the proximity matrix that can be returned using the `proximity=TRUE` option of `randomForest`. Thus the proximity matrix can be taken as a similarity measure, and clustering or multi-dimensional scaling using this similarity can be used to divide the original data points into groups for visual exploration.

We use the `crabs` data in MASS4 to demonstrate the unsupervised learning mode of `randomForest`. We scaled the data as suggested on pages 308–309 of MASS4 (also found in lines 28–29 and 63–68 in '$R_HOME/library/MASS/scripts/ch11.R'), resulting the the `dslcrab` data frame below. Then run `randomForest` to get the proximity matrix. We can then use `cmdscale()` (in package **mva**) to visualize the $1-$proximity, as shown in Figure 3. As can be seen in the figure, the two color forms are fairly well separated.

```
> library(mva)
> set.seed(131)
> crabs.prox <- randomForest(dslcrabs,
+   ntree = 1000, proximity = TRUE)$proximity
> crabs.mds <- cmdscale(1 - crabs.prox)
> plot(crabs.mds, col = c("blue",
+     "orange")[codes(crabs$sp)], pch = c(1,
+     16)[codes(crabs$sex)], xlab="", ylab="")
```

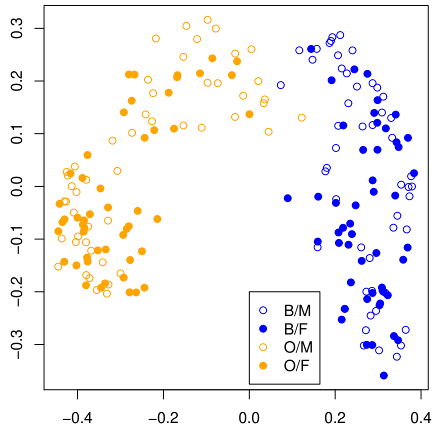

Figure 3: The metric multi-dimensional scaling representation for the proximity matrix of the `crabs` data.

There is also an `outscale` option in `randomForest`, which, if set to `TRUE`, returns a measure of "outlyingness" for each observation in the data set. This measure of outlyingness for the jth observation is calculated as the reciprocal of the sum of squared proximities between that observation and all other observations *in the same class*. The Example section of the help page for `randomForest` shows the measure of outlyingness for the Iris data (assuming they are unlabelled).

Some notes for practical use

- The number of trees necessary for good performance grows with the number of predictors. The best way to determine how many trees are necessary is to compare predictions made by a forest to predictions made by a subset of a forest. When the subsets work as well as the full forest, you have enough trees.

- For selecting $m_{\mathtt{try}}$, Prof. Breiman suggests trying the default, half of the default, and twice the default, and pick the best. In our experience, the results generally do not change dramatically. Even $m_{\mathtt{try}} = 1$ can give very good performance for some data! If one has a very large number of variables but expects only very few to be "important", using larger $m_{\mathtt{try}}$ may give better performance.

- A lot of trees are necessary to get stable estimates of variable importance and proximity. However, our experience has been that even though the variable importance measures may vary from run to run, the ranking of the importances is quite stable.

- For classification problems where the class frequencies are extremely unbalanced (e.g., 99% class 1 and 1% class 2), it may be necessary to change the prediction rule to other than majority votes. For example, in a two-class problem with 99% class 1 and 1% class 2, one may want to predict the 1% of the observations with largest class 2 probabilities as class 2, and use the smallest of those probabilities as threshold for prediction of test data (i.e., use the `type='prob'` argument in the `predict` method and threshold the second column of the output). We have routinely done this to get ROC curves. Prof. Breiman is working on a similar enhancement for his next version of random forest.

- By default, the entire forest is contained in the `forest` component of the `randomForest` object. It can take up quite a bit of memory for a large data set or large number of trees. If prediction of test data is not needed, set the argument `keep.forest=FALSE` when running `randomForest`. This way, only one tree is kept in memory at any time, and thus lots of memory (and potentially execution time) can be saved.

- Since the algorithm falls into the "embarrassingly parallel" category, one can run several random forests on different machines and then aggregate the `votes` component to get the final result.

Acknowledgment

We would like to express our sincere gratitute to Prof. Breiman for making the Fortran code available, and answering many of our questions. We also thank the reviewer for very helpful comments, and pointing out the reference Bylander (2002).

Bibliography

L. Breiman. Bagging predictors. *Machine Learning*, 24 (2):123–140, 1996. 18

L. Breiman. Random forests. *Machine Learning*, 45(1): 5–32, 2001. 18

L. Breiman. Manual on setting up, using, and understanding random forests v3.1, 2002. `http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf`. 18, 19

T. Bylander. Estimating generalization error on two-class datasets using out-of-bag estimates. *Machine Learning*, 48:287–297, 2002. 18, 22

R. Shapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26 (5):1651–1686, 1998. 18

W. N. Venables and B. D. Ripley. *Modern Applied Statistics in S*. Springer, 4th edition, 2002. 19

Andy Liaw
Matthew Wiener
Merck Research Laboratories
`andy_liaw@merck.com`
`matthew_wiener@merck.com`

Some Strategies for Dealing with Genomic Data

by R. Gentleman

Introduction

Recent advances in molecular biology have enabled the exploration of many different organisms at the molecular level. These technologies are being employed in a very large number of experiments. In this article we consider some of the problems that arise in the design and implementation of software that associates biological meta-data with the experimentally obtained data. The software is being developed as part of the Bioconductor project `www.bioconductor.org`.

Perhaps the most common experiment of this type examines a single species and assays samples using a single common instrument. The samples are usually homogeneous collection of a particular type of cell. A specific example is the study of mRNA expression in a sample of leukemia patients using the Affymetrix U95A v2 chips Affymetrix (2001). In this case a single type of human cell is being studied using a common instrument.

These experiments provide estimates for thousands (or tens of thousands) of sample specific features. In the Affymetrix experiment described previously data on mRNA expression for approximately 10,000 genes (there are 12,600 probe sets but these correspond to roughly 10,000 genes). The experimental data, while valuable and interesting require additional biological meta-data to be correctly interpreted. Considering once again the example we see that knowledge of chromosomal location, sequence, participation in different pathways and so on provide substantial interpretive benefits.

Meta-data is not a new concept for statisticians. However, the scale of the meta-data in genomic experiments is. In many cases the meta-data are larger and more complex than the experimental data! Hence, new tools and strategies for dealing with meta-data are going to be needed. The design of software to help manage and manipulate biological annotation and to relate it to the experimental data will be of some importance. As part of the Bioconductor project we have made some preliminary studies and implemented some software designed to address some of these issues. Some aspects of our investigations are considered here.

Issues for consideration

Some of the issues that need to be considered when designing software that associates biological meta-data with experimentally obtained data are listed next. The list is numbered for further reference and the numbering scheme does not reflect importance.

1. the biological meta data are constantly evolving

2. there is some commonality across species

3. there is some commonality across measurement instruments

4. data (either experimental or annotation) may be sensitive and there may be a desire for privacy

5. for most species there is far too much meta-data available to easily provide access to all relevant data without some curation

It is worth distinguishing these meta-data from the meta-data that report information about the samples themselves, the experimental conditions and a variety of other experiment specific data. These data are imporatant and must also be dealt with. For the analysis of microarray data the MIAME standard Brazma et al. (2001) has been proposed and is being widely adopted. These data are being dealt with in the **Biobase** package and are attached directly to the expression data. We now turn our attention to the genomic or biological meta-data for the remainder of this paper.

The design

After careful consideration of the issues raised in Section 8 we settled on a design based on the R package system. The specific design is currently to provide a number of *data* items in the package. For each biological variable of interest (such as chromosome location or LocusLink identifier) a seperate hash table is constructed mapping from the manufacturer's identifier (Affymetrix in our example) to the quantity of interest. These hash tables are then placed in the 'data' directory of the package and documentation for them placed in the 'man' directory. Other software is added as needed.

There are many benefits to supplying meta-data in the form of R packages. These include:

- a version number mechanism

- a mechanism for supplying additional software that may be needed only in specific cases

- mechanisms for documentation (both help pages and vignettes)

- an organizational structure that simplifies automatic generation, distribution and documentation

- mechanisms that support the implementation of quality control processes (the `R CMD` suite of functions)

Given the number of organisms (distinct genomes) and the number of different measurement devices that may be used a robust and reliable indexing and distribution system will be required. Some of these issues have already been addressed within R for distributing packages and that system will be extended to provide support for projects such as the current one.

Since the meta-data are evolving and must be compiled from a variety of sources some form of versioning will be essential. Many biological data resources (such as the Gene Ontology Consortium and GenBank) provide version information that must be propagated to the end user. These data are easily included in the manual page that describes the particular data element in the distributed package. These data specific *build numbers* are essential so that end users can easily compare their results.

One of the major projects undertaken by the Bioconductor project is to enhance the repository management tools that were initially developed by K. Hornik and F. Leisch to provide services through CRAN. The software is currently available in the **reposTools** package available form the Bioconductor anonymous cvs archive. This package represents the collaborative efforts of K. Hornik, F. Leisch, V. Carey, J. Gentry and myself.

From the perspective of biological meta-data the concept of distributing data as R packages via a centralized repository provides many benefits. The data generator (or assembler) has a specific protocol (API) for updating their offerings and a specific mechanism for providing broad access to both new materials and older versions. On the client side, again there is a specific protocol and mechanism for obtaining specific meta-data (generally the most current). The processes that the client must carry out are entirely moderated through R. While other alternatives are possible it seems that the end-user will find it easier if they have a single system to learn for both data management and data analysis.

Among the objectives of the repository project are to enhance the version management (users may want to maintain multiple versions of particular libraries) capabilities of R, to enable the use of multiple repositories as suppliers of software and to allow users to become providers of software.

Once **reposTools** is included in R users will interact with the software through `install.packages` and `update.packages`. In the current implementation these have a `2` appended to keep them distinct from the system versions. **reposTools** contains vignettes that will help guide interested readers through the steps needed to set up their own repository and to export it. More details on this project will be presented in later editions of R News.

Another important component of the approach taken is the assumption that the meta-data packages will need to be automatically generated and that there must be a mechanism that allows them to be updated. Autogeneration of packages is the only practical method for dealing with the large number of species and of technologies that need to be accomodated.

To facilitate this process we have developed the **AnnBuilder** package. The basic philosophy embodied in this package is to allow the package builder to supply the locations of a set of source data files. From these different variables are extracted and assembled into a large coherent set of data files. In this article we present an overview of this process and we refer the reader to Zhang et al. (2002, in preparation) for specific details.

In essence **AnnBuilder** supports the construction and distribution of custom annotation packages. The user specifies the locations of data resources, the mechanisms for extracting and combining the data and finally the required output. Specification of data resources can be URLs or local files. The user must also specify which fields should be extracted from which packages.

The data processing part of **AnnBuilder** extracts the specified fields from each file. The output may be to a database or in some cases to a file. The main purpose of this step is often data reduction. The primary source data is often much larger than is needed for any particular experiment. These tables are then processed (using a variety of tools including SQL and R) and the resultant unified mapping table can be exported as an R package, an XML data file (the DTD is) or as a database file that is suitable for querying by any means (including R, Perl, Python etc).

When multiple mappings or sources are available for a particular data item some form of reduction is needed. Reasonable suggestions include using the most common value (mode), nominating one source as trusted and using it if there are differences. **AnnBuilder** provides a general mechanism that allows the user to associated a SQL query statement with each variable to perform the selection. In most cases the function is simply the identity (since we have only one source for many of the quantities).

In many ways this sort of data assemblage is often required in other fields. There is nothing about **AnnBuilder** that is specific to biological data and it may be beneficial to consider the application of **AnnBuilder** in those areas. We would also like to consider slightly different models for data acquisition. As web-services become more common it will be important to extend the **AnnBuilder** concept to deal with data resources of this type.

For specific examples of the output of **AnnBuilder** the reader is referred to the Bioconductor Web site. Many different data packages are available by simply following the links. Interactions with the various Bioconductor packages are primarily mediated by the **annotate** package.

Discussion

We have considered some approaches to dealing with data complexity. By abstracting out both the platform (technology) and the species we have developed an annotation system that is practical and that can be maintained and extended.

While we have tried to make the data processing automatic it is not. The data are sufficiently complex and different to prevent complete automation. However, the data assemblage process is largely automatic and well documented. The requirements on both the server side and the client side are reasonably explicit and will be further refined as the projects evolve. We believe our efforts constitute a reasonable solution and have been using this methods for some time now.

The distribution of the resultant data is somewhat easier. Once appropriate names have been selected for the data packages they can easily be distributed and updated using R's repository system. The software in supplied in the **reposTools** package will need some testing but should provide an adequate basis for the distribution of these data packages.

By assembling data packages we also define an interface that can be used by other packages to access that data. In particular the functions in the package **annotate** are designed to be applied to any conforming experimental data and any conforming annotation package. The requirements placed on both the experimental data and the annotation package are documented and any package or data source that satisfies those requirements can be used.

It is rather interesting to close by reviewing some of the issues that have been raised in developing this paradigm.

- using http connections to access data on the web
- using DBI to interact with the data base that will do the heavy duty processing
- using XML to process the data
- automatic package generation
- using R resources to build in quality control procedures

We finish our discussion by reviewing the issues raised in Section 8. By using R packages we address issue 1 since the package system allows us to version and update the data. The **AnnBuilder** approach addresses issues 2, 3 and 5. Data reduction is under the control of the data assembler. The notion of a package that generates packages is our method of dealing specifically with issues 2 and 3. The repository mechanism allows the data assembler to distribute the assembled data (they may choose to restrict distribution only to specific sites).

Acknowledgment

Robert Gentleman's work is supported in part by NIH/NCI Grant 2P30 CA06516-38.

Bibliography

Affymetrix. *Affymetrix Microarray Suite User Guide*. Affymetrix, Santa Clara, CA, version 5 edition, 2001. 22

A Brazma, P Hingamp, and J Quackenbush. Minimum information about a microarray experiment: towards standards for microarray data. *Nature Genetics*, 29:365–371, 2001. 23

Jianhua Zhang, Vincent Carey, and Robert Gentleman. An extensible application for assembling annotation for genomic data. *Bioinformatics*, 19, 2002. 24

Jianhua Zhang, Vincent Carey, A. J. Rossini, and R. C. Gentleman. Annbuilder - an open source application for genomic data annotation. *JSS*, in preparation. 24

Robert Gentleman
DFCI
rgentlem@jimmy.harvard.edu

Changes to the R-Tcl/Tk package

by Peter Dalgaard

Introduction

In a previous issue of R News, I gave a small tutorial on the basic use of the `tcltk` package for creating graphical user interfaces (Dalgaard, 2001). Since then, there has been a number of changes to the package, and the purpose of this paper is to outline the new possibilities offered by the modified interface.

Some of the changes were incompatible with old code. In particular, some of the examples in Dalgaard (2001) no longer run. The last section of the paper contains a revised version of the scripting widget.

Control variable changes

The old-style interface to Tcl variables allowed you to change the Tcl variable `foo` using syntax like

```
tclvar$foo <- "Hello, World"
```

With this approach, `foo` becomes a global Tcl variable, and there is no way to ensure that two R functions do not accidentally use the same variable name. (This is less of a problem in Tcl because there you can use techniques like prefixing the variable with the window name.)

Therefore, a new technique is introduced using the object class `"tclVar"`. The creator function for that class is `tclVar()` and it works by creating a new Tcl variable. You generally do not need to care about the names on the Tcl side, but they are simply `::RTcl1`, `::RTcl2`, and so forth (the `::` prefix ensures that they are in Tcl's global namespace). The accessor function is `tclvalue()` and a typical usage is like this:

```
foo <- tclVar()
tclvalue(foo) <- "Hello, World"
```

Notice that you have to create the variable before it can be used.

The `"tclVar"` objects are subject to finalization, so that when they go out of scope, the garbage collector will eventually remove their Tcl counterparts.

Tcl objects

The C interface to Tcl contains the notion of *dual ported* objects. All Tcl objects have a *string representation*, but they can also have a "dual personality" as doubles, integers, or lists of strings, doubles, or integers. (There are other possibilities but we ignore them here.)

This allows you to access Tcl variables without going via the string representation. The latter can be a major pain because of "quoting hell": The need to escape certain characters because they otherwise have special meaning to the Tcl interpreter.

The interface from R maps double, integer, and character vectors to Tcl lists of the corresponding object types. This works via an accesor function called `tclObj()`. An example should convey the gist of the interface:

```
> foo <- tclVar()
> tclObj(foo) <- c(pi,exp(1))
> tclvalue(foo)
[1] "3.14159265359 2.71828182846"
> as.character(tclObj(foo))
[1] "3.14159265359" "2.71828182846"
> as.double(tclObj(foo))
[1] 3.141593 2.718282
> as.integer(tclObj(foo))
[1] NA NA
```

Notice that the elements can not be interpreted as integers, and that `as.integer` in that case returns `NA` rather than attempting to truncate the values.

The return value from `tclObj()` is an object of class `"tclObj"`. It is only possible to modify such objects when there is an underlying Tcl variable. Also, the only way to modify a Tcl object is to extract it in its entirety, modify the R representation, and store it back into the Tcl variable.

The issue that prompted the development of this interface was the handling of listboxes. Using Tcl objects, this can be done as simply as

```
mylist <- tclVar()
tkpack(lb <- tklistbox(tt <- tktoplevel(),
                       listvariable = mylist))
tclObj(mylist) <- month.name
```

Using the string representation, you would need something like

```
tclvalue(mylist) <-
    paste(month.name, collapse=" ")
```

which may not look too bad, but if you need to have a list elements with spaces inside things get complicated.

With the Tcl object interface, you have a direct connection to the list contents, and can do things like

```
as.character(tclObj(mylist))[
    as.integer(tkcurselection(lb))+1]
```

which will return the text of any selected items. Notice that it is necessary to add 1 since Tcl uses zero-based indexing.

Return values

The return values from `.Tcl` calls (and thus most of the interface functions, the main exception being the widget creation commands) are now objects of class `"tclObj"` instead of a string value. This is useful for much the same reasons as described in the previous section.

For instance, if you turn on multiple selection mode for a listbox with

```
tkconfigure(lb, selectmode="multiple")
```

then the string representation of

```
tkcurselection(lb)
```

might be `"0 3 4 5"` and it would be the programmer's responsibility to parse the string into four integers. With the object representation, you push that responsibility into the Tcl layer and it becomes possible to access the vector of indices directly using `as.integer(tkcurselection(lb))`.

Changing the return value created some incompatibilities with some earlier code. Fortunately the extent of the problems is limited since return values are often ignored, and you can alway restore compatibility by taking `tclvalue` of the returned object.

Callback changes

Several new formats have been introduced for specifying callbacks. Some of these just offer notational convenience, but some were necessary because the old interface had no way to invoke Tcl's `break` command. For instance you bind an action to, say, `<Control-Return>` then any other bindings for `<Return>` will also execute, unless `break` is used. This can now be coded as

```
tkpack(txt <- tktext(tktoplevel()))
tkbind(txt, "<Control-Return>",
       expression(print(tkget(txt,"0.0","end")),
                  break))
```

Compared to the old style where a callback had to be a function, the new rules allow some simplification when there are no arguments. E.g.,

```
but <- tkbutton(tt, text="Hit me!",
                command=expression(cat("OW!!")))
```

Conversely, if you do need to be able to pass arguments to the callback *and* need to use `break`, then you can do so by specifying a function for one or more elements of the callback expression. That is, you could use

```
tkbind(txt, "<Control-Button-1>",
       expression(function(x,y) cat(x,y,"\n"),
                  break))
```

to create a callback that prints out the mouse cursor position, without performing the text cursor motion bound to `<Button-1>`.

New version of the script widget

The script-window application in Dalgaard (2001) got hit rather badly by the interface changes. Below is a version that works with the new interface.

Notice that it is necessary to insert `tclvalue()` constructions in several places, even when the return values are only used as arguments to Tcl/Tk routines. You can sometimes avoid this because the default treatment of arguments (in `.Tcl.args()`) is to preprocess them with `as.character()`, but for objects of class `"tclObj"` this only works if there are no whitespace characters in the string representation. The contents of the script window and the files that are read can obviously contain spaces and it is also not safe to assume that file names and directory names are single words.

Notice also that several new functions have been added to the interface, `tkopen`, `tkclose`, `tkfile.tail`, etc. (Beware that `tkread` was not added until R version 1.6.1; for version 1.6.0 you need `tkcmd("read",...)` instead.)

```
tkscript <- function() {
  wfile <- ""
  tt <- tktoplevel()
  txt <- tktext(tt, height=10)
  tkpack(txt)
  save <- function() {
    file <- tclvalue(tkgetSaveFile(
      initialfile=tclvalue(tkfile.tail(wfile)),
      initialdir=tclvalue(tkfile.dir(wfile))))
    if (!length(file)) return()
    chn <- tkopen(file, "w")
    tkputs(chn, tclvalue(tkget(txt,"0.0","end")))
    tkclose(chn)
    wfile <<- file
  }
  load <- function() {
    file <- tclvalue(tkgetOpenFile())
    if (!length(file)) return()
    chn <- tkopen(file, "r")
    tkinsert(txt, "0.0", tclvalue(tkread(chn)))
    tkclose(chn)
    wfile <<- file
  }
  run <- function() {
    code <- tclvalue(tkget(txt,"0.0","end"))
    e <- try(parse(text=code))
    if (inherits(e, "try-error")) {
      tkmessageBox(message="Syntax error",
                   icon="error")
      return()
    }
    cat("Executing from script window:",
        "-----", code, "result:",  sep="\n")
    print(eval(e))
  }
  topMenu <- tkmenu(tt)
  tkconfigure(tt, menu=topMenu)
  fileMenu <- tkmenu(topMenu, tearoff=FALSE)
  tkadd(fileMenu, "command", label="Load",
        command=load)
  tkadd(fileMenu, "command", label="Save",
        command=save)
  tkadd(topMenu, "cascade", label="File",
        menu=fileMenu)
  tkadd(topMenu, "command", label="Run",
        command=run)
}
```

Future developments

The **tcltk** package is still nowhere near its final state, and there are several changes and enhancements in the works.

The use of Tcl objects has proven to be a big advantage, and I'd like to extend its use so that we as far as possible can avoid direct use of string representations. In particular, I believe that it should be possible to modify the current `.Tcl` interface to use Tcl objects and thereby avoid the "quoting hell" associated with passing arbitrary character strings. Also, it ought to be possible to pass Tcl objects directly to Tcl commands, which would remove the need for many calls to `tclvalue`.

We need to become able to take advantage of the various extensions that exist to Tcl and Tk. Sometimes this is almost trivial, but in other cases attempting to do so reveals shortcomings of the current interface. For instance, it is tempting to use the features of the TkTable widget to implement a better spreadsheet-like data editor than the raw X11 one we currently have. However, this entails finding a way to return a value from a callback, and/or an extension of the interface to Tcl variables that can deal with arrays. Notice that it is not quite trivial to return values from a callback, since we cannot in general expect to convert an arbitrary R object to something that makes sense in Tcl.

Another thing that I have been working on is the addition of a Tk-based console for R. The main reason is that you cannot add menus to a terminal window, and a free-floating toolbar would probably not be too user-friendly. R-1.6.0 and later (Unix versions) have a couple of stubs to allow redirection of input and output to Tcl functions, and I have an as yet unpublishable sketch of the rest.

Finally, there is the issue of writing a graphics driver. Luke Tierney has a fairly decent solution in the **tkrplot** package, but it would be interesting to incorporate more features of the Tk canvas, which does allow some manipulations that you don't have with standard devices. E.g., you can support simple interactive graphics by tagging plot elements, so that you could move or delete them afterwards. Unfortunately, the Tk canvas is limited with respect to clipping and rotation of text, but the Zinc extension (`http://www.openatc.org/zinc`) looks promising.

Bibliography

Peter Dalgaard. A primer on the R-Tcl/Tk package. *R News*, 1(3):27–31, September 2001. URL `http://CRAN.R-project.org/doc/Rnews/`. 25, 26

Peter Dalgaard
Department of Biostatistics
University of Copenhagen, Denmark
`p.dalgaard@biostat.ku.dk`

Sweave, Part I: Mixing R and LaTeX

A short introduction to the Sweave file format and corresponding R functions

by Friedrich Leisch

This is the first article in a two part mini series on Sweave (Leisch, 2002), a tool that allows to embed the R code for complete data analyses in LaTeX documents. In this issue we will introduce the Sweave file format and R functions to process it, and demonstrate how to use Sweave as a reporting tool for literate statistical practice (Rossini, 2001). The companion article scheduled for the next issue of R News will concentrate on how to use files in Sweave format to write primers or manuals for R packages that can be automatically checked for syntax errors in the code or inconsistencies between examples and implementation.

The traditional way of writing a report as part of a statistical data analysis project uses two separate steps: First, the data are analyzed, and afterwards the results of the analysis (numbers, graphs, ...) are used as the basis for a written report. In larger projects the two steps may be repeated alternately, but the basic procedure remains the same. R supports this in a number of ways: graphs can be saved as PDF, EPS, or WMF which in turn can be included in LaTeX or Word documents. LaTeX tables can be created by specifying the columns and row separators in `write.table()` or using the package **xtable**. The basic paradigm is to write the report around the results of the analysis.

The purpose of Sweave is to create dynamic reports, which can be updated automatically if data or analysis change. Instead of inserting a prefabricated graph or table into the report, the master document contains the *R code* necessary to obtain it. When run through R, all data analysis output (tables, graphs, ...) is created on the fly and inserted into a final LaTeX document. The report can be automatically updated if data or analysis change, which allows for truly reproducible research.

A small example

Sweave source files are regular noweb files (Ramsey, 1998) with some additional syntax that allows control over the final output. Noweb is a simple literate programming tool which allows to combine program source code and the corresponding documentation into a single file. These consist of a sequence of code and documentation segments, called *chunks*. Different command line programs are used to extract the code (*"tangle"*) or typeset documentation together with the code (*"weave"*).

A small Sweave file is shown in Figure 1, which contains four code chunks embedded in a simple LaTeX document. '`<<...>>=`' at the beginning of a line marks the start of a code chunk, while a '`@`' at the beginning of a line marks the start of a documentation chunk. Sweave translates this into a regular LaTeX document, which in turn can be compiled by `latex` to Figure 2.

The code chunks

The main work of Sweave is done on the code chunks. All code chunks are evaluated by R in the order they appear in the document[1]. Within the double angle brackets we can specify options that control how the code and the corresponding output are rendered in the final document. The first code chunk (lines 5–8 in Figure 1) declares that neither the R code (`echo=false`) nor output (`results=hide`) shall be included. The purpose of this chunk is to initialize R by loading packages and data, we want to hide these technical details from the reader.

Let us skip the text in lines 10–19 for the moment and go directly to the next code chunk in lines 20–22. It uses the default settings for all options (nothing is specified within the double angle brackets): both input and output are shown to the user (see Figure 2), the chunk is rendered such that it emulates the R console when the code is typed at the prompt. All input and output are automatically encapsulated in verbatim-like environments.

The next code chunk can be found at lines 30–31. It uses the package **xtable** to pretty-print the coefficient matrix of the linear regression model. By specifying `results=tex` we tell Sweave that the output of this code chunk is regular TeX code and hence needs no protection by a verbatim environment.

The last code chunk in lines 36–38 is marked as a figure chunk (`fig=true`) such that Sweave creates EPS and PDF files corresponding to the plot created by the commands in the chunk. Furthermore, an `\includegraphics{}` statement is inserted into the LaTeX file. Options `width` and `height` are passed to R's graphics devices and determine the size of the figure in the EPS and PDF files.

In line 28 we use `\SweaveOpts{echo=false}` to modify the default for option `echo` to the value of `false` for all code chunks following, hence the code for the last two chunks is not shown in Figure 2. It has exactly the same effect as if we had included `echo=false` within the double angle brackets of the two chunks.

[1] There are ways to suppress evaluation or re-use chunks, which is beyond the scope of this article.

```
\documentclass[a4paper]{article}

\begin{document}

<<echo=false,results=hide>>=
library(lattice)
library(xtable)
data(cats, package="MASS")
@

\section*{The Cats Data}

Consider the \texttt{cats} regression example from Venables \& Ripley
(1997). The data frame contains measurements of heart and body weight
of \Sexpr{nrow(cats)} cats (\Sexpr{sum(cats$Sex=="F")} female,
\Sexpr{sum(cats$Sex=="M")} male).

A linear regression model of heart weight by sex and gender can be
fitted in R using the command
<<>>=
lm1 = lm(Hwt~Bwt*Sex, data=cats)
lm1
@
Tests for significance of the coefficients are shown in
Table~\ref{tab:coef}, a scatter plot including the regression lines is
shown in Figure~\ref{fig:cats}.

\SweaveOpts{echo=false}

<<results=tex>>=
xtable(lm1, caption="Linear regression model for cats data.", label="tab:coef")
@

\begin{figure}
  \centering
<<fig=true,width=12,height=6>>=
lset(col.whitebg())
print(xyplot(Hwt~Bwt|Sex, data=cats, type=c("p", "r")))
@
  \caption{The cats data from package MASS.}
  \label{fig:cats}
\end{figure}

\end{document}
```

Figure 1: A minimal Sweave file: `example.Snw`.

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2.9813	1.8428	1.62	0.1080
Bwt	2.6364	0.7759	3.40	0.0009
SexM	−4.1654	2.0618	−2.02	0.0453
Bwt:SexM	1.6763	0.8373	2.00	0.0472

Table 1: Linear regression model for cats data.

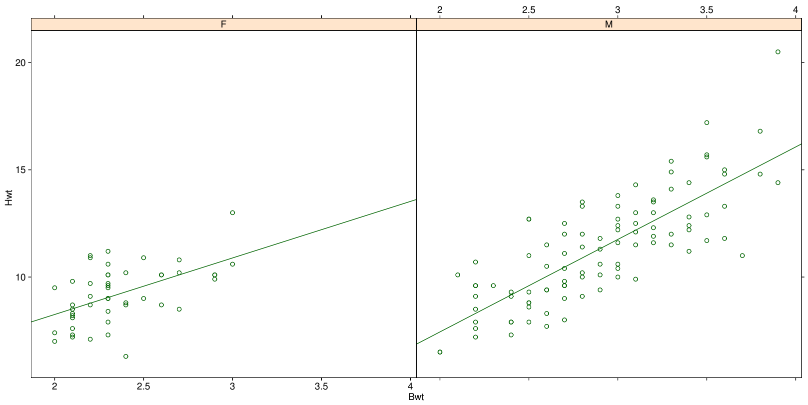

Figure 1: The cats data from package MASS.

The Cats Data

Consider the `cats` regression example from Venables & Ripley (1997). The data frame contains measurements of heart and body weight of 144 cats (47 female, 97 male).

A linear regression model of heart weight by sex and gender can be fitted in R using the command

```
> lm1 = lm(Hwt ~ Bwt * Sex, data = cats)
> lm1

Call:
lm(formula = Hwt ~ Bwt * Sex, data = cats)

Coefficients:
(Intercept)          Bwt         SexM     Bwt:SexM
      2.981        2.636       -4.165        1.676
```

Tests for significance of the coefficients are shown in Table 1, a scatter plot including the regression lines is shown in Figure 1.

Figure 2: The final document is created by running `latex` on the intermediate file 'example.tex' created by `Sweave("example.Snw")`.

Using S objects in text

Let us now return to the text paragraph in lines 13–16. It contains three `\Sexpr{}` statements. Sweave replaces them by the value of the corresponding S expression, which should be a simple character string (or something that can be coerced to a string by `as.character()`). In the example we use it to avoid hard-coding the size of the data set. If the number of observations changes we do not need to change anything in our Sweave file, we simply re-run `Sweave()` and `latex` and the report is up-to-date.

Writing Sweave files

The Emacs text editor offers a perfect authoring environment for Sweave, especially for people who already use Emacs for writing LaTeX documents and interacting with R. ESS (*Emacs speaks statistics*, Rossini et al., 2003) allows to connect an Sweave file to a running R process while writing the document. Code chunks can be sent to R and evaluated using simple keyboard shortcuts or popup menus. Syntax highlighting, automatic indentation and keyboard shortcuts depend on the location of the pointer: in documentation chunks Emacs behaves as if editing a standard LaTeX file, when the pointer moves to a code chunk the mode switches automatically to S programming.

However, it is not necessary to use Emacs, Sweave is a standalone system, the noweb source files for Sweave can be written using any text editor. Even the noweb syntax in not a necessity, because Sweave is highly configurable. Currently there are two syntaxes available, the noweb syntax described above and a LaTeX-based syntax. In LaTeX syntax the first code chunk of the example looks like

```
\begin{Scode}{echo=false,results=hide}
  library(lattice)
  library(xtable)
  data(cats, package="MASS")
\end{Scode}
```

Processing Sweave files

Sweave is contained in the standard R package **tools** (R version 1.5.0 or higher). The Sweave file 'example.Snw' can be processed using the R commands

```
> library(tools)
> Sweave("example.Snw")
Writing to file example.tex
Processing code chunks ...
 1 : term hide
 2 : echo term verbatim
 3 : term tex
 4 : term verbatim eps pdf

You can now run LaTeX on example.tex
```

Sweave shows a status line per code chunk indicating which options are active. The companion command

```
R> Stangle("example.Snw")
Writing to file example.R
```

can be used to extract the code of all chunks into an R source file.

Resources and summary

The example Sweave file used in this article can be found as 'example-3.Snw' at the Sweave homepage http://www.ci.tuwien.ac.at/~leisch/Sweave, where you also find a manual and more examples.

Sweave is already used for a wide variety of applications: Reports for medical statistical consulting that can be updated automatically when new data arrive or data change; lecture notes for statistics classes with executable examples; and manuals with embedded examples for R packages that can tested as part of the `R CMD check` suite, so-called package *vignettes*. The last application will be the topic of the second part of this article in the next issue of R News.

Bibliography

F. Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Härdle and B. Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physika Verlag, Heidelberg, Germany, 2002. URL http://www.ci.tuwien.ac.at/~leisch/Sweave. ISBN 3-7908-1517-9. 28

N. Ramsey. *Noweb man page*. University of Virginia, USA, 1998. URL http://www.cs.virginia.edu/~nr/noweb. version 2.9a. 28

A. Rossini. Literate statistical analysis. In K. Hornik and F. Leisch, editors, *Proceedings of the 2nd International Workshop on Distributed Statistical Computing, March 15-17, 2001, Technische Universität Wien, Vienna, Austria*, 2001. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2001/Proceedings/. ISSN 1609-395X. 28

A. J. Rossini, R. M. Heiberger, R. Sparapani, M. Mächler, and K. Hornik. Emacs speaks statistics: A multiplatform, multi-package development environment for statistical analysis. *Journal of Computational and Graphical Statistics*, 2003. (Accepted for publication). 31

Friedrich Leisch
Department of Statistics & Decision Support Systems
University of Vienna, Austria
leisch@R-project.org

R Help Desk

Automation of Mathematical Annotation in Plots

Uwe Ligges

Welcome to the R Help Desk

Welcome to the first issue of a regular series of *R Help Desk* columns.

As the title of the column suggests, it is intended to present answers to frequently asked questions related to R, for example well known questions from the R mailing lists. More specifically, the intention is to address problems which cannot be described and explained completely in a few lines of text, as it is common in manuals, help pages, typically styled answers on mailing lists, or the R FAQ (Hornik, 2002).

So, on the one hand, articles published in this column are intended to present solutions to common problems. On the other hand, to be easy to read by less experienced R users, the articles should not be too technical.

Contributions

It is a pleasure to start as the editor of this column. Like Bill Venables in his first issue of the *Programmer's Niche*, I would like to take the opportunity to invite *you*, the reader, to *contribute* articles. If you have any ideas on how to describe solutions to common programming problems and (more or less) frequently asked questions, please send your contributions to `ligges@statistik.uni-dortmund.de`.

Introduction to mathematical annotation in plots

Many users know about R's capabilities of typesetting mathematical annotation in plots, which were introduced by Murrell and Ihaka (2000). Related to this topic, I frequently heard and read sentences like "Mathematical annotation in plots can be typeset using a LaTeX–like syntax".

- This statement is partly true for two reasons: Typesetting is programmed in a way, both in R and in LaTeX. The "keywords" for typesetting objects like greek letters, fractions etc. are quite similar.

- The statement is mainly wrong, or at least confusing: The *syntax* is fortunately more or less the syntax of the S language with some small specialities, therefore most R users will know about its main rules.

Let us start collecting the required information to typeset formulas in R. We do not need any special functions to typeset, since the regular mechanisms used to typeset character strings, like the arguments `main` or `xlab` in `plot()`, or functions like `text()` etc., are sufficient.

We will not be able to specify mathematical annotation as character strings, but we need to specify them as S expressions or calls — without evaluating them. Some functions to specify unevaluated S expression, or, more specifically, functions to manipulate and work with S expressions and calls are described in detail by Venables and Ripley (2000). For those interested in more technicalities, the Programmer's Niche by Venables (2002) in the previous newsletter gives some nice insights into the language.

Anyway, the most frequently used reference for our purpose certainly is the help page `?plotmath`.

Having collected most of the required information, we know that `expression()` is an appropriate function to specify an unevaluated S expression. Thus, we can easily produce the following example (just try it out!).

```
> plot(0, main =
    expression(y == alpha*x[1] + beta*x[2]^2))
```

The resulting plot will have a rather nice formula in its main title.

Automation

The question how to replace some variables in *formulas* by their *values* seems to be more sophisticated, but is still documented in the examples of `?plotmath`.

In particular, the function `substitute()` is designed to substitute any variables in a call by their value (from a given environment or list of objects). As an extension of our first example

```
> a <- 3.5
> x <- 1:2
> substitute(y == a + alpha*x[1] + beta*x[2]^2,
    list(a = a))
```

will replace the variable `a` (but not `x`) in the expression with its value. Such a mechanism is of special interest for some automated generation of plots, where the user is not willing to specify the calls to each plot separately.

Let us construct a small example (you might want to try it out before looking at the code): Consider you are working with a bivariate normal distribution, for which you automatically calculate the mean μ and the covariance matrix Σ_x:

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \quad \Sigma_x = \begin{pmatrix} \sigma_1 & \sigma_3 \\ \sigma_2 & \sigma_4 \end{pmatrix}.$$

In the same procedure, you want to generate a nice plot for some presentation, including the formula for the density of your multivariate normal distribution:

$$f(x) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma_x)}} \times \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma_x^{-1}(x-\mu)\right).$$

Further on, suppose you would like to print out the calculated values of μ and Σ_x in the same plot. One possible solution would be the following code.

```
> ## Let us set up an empty plot:
> plot(1:8, type = "n")
> ## a list of imaginary calculated values:
> param.list <- list(mu1 = 0, mu2 = 0,
    s1 = 3, s2 = 2, s3 = 2, s4 = 4)
> ## typeset density of 2-var. normal dist.
> text(1, 6, adj = 0, labels = expression(
    f(x) == frac(1, sqrt((2 * pi)^n ~~
      det(Sigma[x]))) ~~ exp * bgroup("(",
      -frac(1, 2) ~~ (x - mu)^T * Sigma[x]^-1 *
      (x - mu), ")")))
> ## typeset concrete values of mu and Sigma
> ##  (from param.list):
> text(8, 3, adj = 1, labels = substitute(
    "with " * mu == bgroup("(", atop(mu1, mu2),
      ")") * " , " * Sigma[x] ==
      bgroup("(", atop(s1 ~~ s3, s2 ~~ s4), ")"),
      param.list))
```

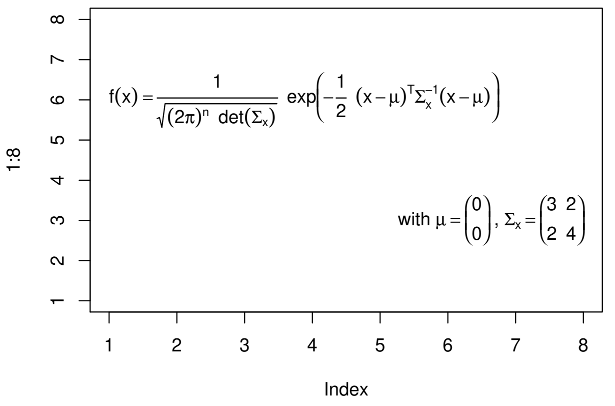

Figure 1: Example – Typesetting the density of a bivariate normal distribution with substituted values

More automation

From another point of view, substitution might be desirable for variables (objects) *containing* expressions, or expression-like strings. Consider you would like to pass such an object as an argument to a function, and within that function the title for the plot shall be constructed from different elements.

The following function will plot the object x according to its class, and label the plot with a compounded title. For the labelling two different objects `arg2` (an expression) and `arg3` (a character string) are expected by the function. This particular design of the function is chosen for illustrating two possible ways to achieve the typesetting:

```
> my.foo <- function(x, arg2, arg3, ...){
    arg3 <- parse(text = arg3)[[1]]
    plot(x, main =
      substitute("Formula in \'arg2\': " * arg2
        * ";  Formula in \'arg3\': " * arg3,
        list(arg2 = arg2, arg3 = arg3)),
      ...)
  }
> my.foo(1:10, arg2 = quote(alpha[1] == 5),
    arg3 = "y == alpha + beta*x + epsilon")
```

Neither for `arg2`, nor for `arg3`, it is possible to substitute the variable by an object of mode `expression`, which can include several objects of mode `call`, because `substitute()` will fail in that case. Instead, the trick is to pass an object of mode `call`. In the first case, `arg2` is specified in the function call using `quote()`. In the second case, `arg3` is specified as a character string that is parsed (i.e. an expression is returned) inside the function. In order to get an object of mode `call`, only the first element of the returned list is extracted.

Legends

Another quite frequently asked question related to mathematical annotation is how to deal with a couple of formulas at once, as required in legends. If substitution of variables by values is not necessary, `expression()` will still do the trick:

```
> plot(1:8, type = "n")
> legend(2, 3, expression(alpha^2, x[5], Omega))
```

But what about substituting? Consider you have calculated values $\alpha = 1, \beta = 2$, and want to present those values within any legend. You will have to substitute the variables of each legend's element separately before putting them together in an expression. The latter can be done by `do.call()` in a somewhat tricky manner, constructing a call to `expression()` with the legend's elements as its arguments:

```
> a <- 3; b <- 5
> legend1 <- substitute(alpha == a, list(a = a))
> legend2 <- substitute(beta == b, list(b = b))
> legend(5, 5,
    do.call("expression", list(legend1, legend2))
```

Working intensively with mathematical annotation in plots involves the use of expressions and calls, thus it is close to the language. Therefore this first issue of the *R Help Desk* got a bit more technical than it was intended to be.

I would like to close with a nice citation of Venables (2002): "Mind Your Language".

Bibliography

K. Hornik (2002). *The R FAQ*. ISBN 3-901167-51-X, http://www.ci.tuwien.ac.at/~hornik/R/. 32

P. Murrell and R. Ihaka (2000). An Approach to Providing Mathematical Annotation in Plots, *Journal of Computational and Graphical Statistics*, 9(3): 582–599. 32

W. N. Venables and B. D. Ripley (2000). *S Programming*. Springer-Verlag, New York. 32

W. N. Venables (2002). Programmer's Niche, *R News*, 2(2): 24–26, ISSN 1609-3631, http://CRAN.R-project.org/doc/Rnews/. 32, 33

Uwe Ligges
Fachbereich Statistik, Universität Dortmund, Germany
liggesstatistik.uni-dortmund.de

Changes in R

by the R Core Team

User-visible changes

- The default colour palette now has `"grey"` instead of `"white"` in location 8. See `palette()`.
- `grid(nx)` behaves differently (but the same as in R versions <= 0.64).

New features

- `barplot()` has a new argument 'axis.lty', which if set to 1 allows the pre-1.6.0 behaviour of plotting the axis and tick marks for the categorical axis. (This was apparently not intentional, but `axis()` used to ignore lty=0.) The argument 'border' is no longer "not yet used".
- New operator `::` in the grammar, for name spaces.
- New faster `rowsum()`, also works on data frames.
- `grep()`, `sub()`, `gsub()` and `regexpr()` have a new argument 'perl' which if `TRUE` uses Perl-style regexps from PCRE (if installed). New capabilities option `"PCRE"` to say if PCRE is available.
- Preparations for name space support:
  - Functions in the **base** package are now defined in a name space. As a temporary measure, you can disable this by defining the environment variable `R_NO_BASE_NAMESPACE`.
  - `UseMethod` dispatching now searches for methods in the environment of the caller of the generic function rather than the environment where the generic is defined.
- The objects created in the **methods** package to represent classes, generic functions, method definitions, and inheritance relations now themselves belong to true classes. In particular, the `"classRepresentation"` objects follow the description in "Programming with Data" (section 7.6).
- Other additions and changes to the **methods** package:
  - The function `setOldClass()` has been added, following the description on page 450 of "Programming with Data". Use it if old-style classes are to be supplied in signatures for `setMethod`, particularly if the old-style classes have inheritance. Many of the old-style classes in the base package should be pre-specified; try `getClass("mlm")`,e.g.
  - The `setGeneric()` function applies some heuristics to warn about possibly erroneous generic function definitions. (Before, obscure bugs could result.)
  - The function `promptMethods()` has been revised to work better and to provide aliases for individual methods.
  - The behavior of the `as()` function has been generalized, in particular with a '`strict=`' argument, the general goal being to let simple extensions of classes pass through in method dispatch and related computations without altering the objects. More to make method behavior more "natural" than for direct use.
  - Some inconsistencies following `detach("package:methods")` have been removed, so it *should* be possible to detach/re-attach the methods package.
- New methods (`[[`, `print`, `str`) and extended `plot()` method (including logical 'horiz') for `"dendrogram"` class.

- `sprintf()` now checks the agreement between formats and object types, and handles special values (`NA`, `Inf`, ...) correctly.

- `chol()` now uses a tolerance for non-positive-definiteness and so should give more consistent results across platforms.

- New function `agrep()` for approximate (fuzzy) string matching.

- `help.search()` can now use both approximate (fuzzy) and regular expression matching. By default, if the pattern to be matched consists of only alphanumeric characters, whitespace or a dash, approximate matching is used.

- `axis()` has three new optional arguments 'col', 'lty', and 'lwd' all for drawing the axis line and tick marks.

- Function `vcov()` (formerly in **MASS**), a generic function to return the variance-covariance matrix of the parameter estimates of a fitted model.

- `duplicated()` and `unique()` have methods for matrices and arrays (based on ideas from Jens Oehlschlägel).

- Internally memory sizes and counts of cons cells are now stored in unsigned longs. This allows memory limits to be set and objects created in the range 2-4Gb on 32-bit platforms, and allows 64-bit platforms to use much larger amounts of memory.

- Command-line flags to set memory can now use the suffix '`G`' for gigabytes. The setting of maximum vsize is now only limited by the platform's address space.

- All warning and error messages are truncated to a length set by `options(warning.length=)`, defaulting to 1000. (Previously most (but not quite all) were truncated at 8192 characters.)

- `[dpqr]gamma()` check for shape parameter > 0.

- `as.POSIX[cl]t()` can now convert logical `NA`s.

- All installed packages (even those shipped with R) are given a '`Built`' field in the '`DESCRIPTION`' file.

- `as.data.frame()` now coerces logical matrices into logical columns (rather than factors).

- `[[<-.data.frame` no longer coerces character replacement values to factor. This is consistent with using '`$`' to replace and with S4.

- `library()` attempts to detect improperly installed packages, so as from this version an installed package must have a '`DESCRIPTION`' file and that file must have been stamped with a '`Built:`' line (which was introduced in 1.2.0). Under Unix-alikes, the platform is checked against that used for installation.

- `print.factor()` has new arguments 'max.levels' (with a smart default) and 'width'. `print.ordered()` is no longer needed.

- `RNGkind()` has an additional option for normal random generators: `"Inversion"`.

- `data.frame()` recycles factors and `"AsIs"` objects as well as atomic vectors.

- `predict.lm()` warns if 'newdata' is supplied and the fit was rank-deficient, as this can be misleading.

- `rect()` accepts additional graphics parameters through a '`...`' argument (in the same way as polygon).

- `strwidth()` and `strheight()` now coerce their first argument in exactly the same way `text()` does, so a wider range of inputs is allowed.

- `prompt()`'s default and data.frame methods have a new 3rd argument 'name' allowing them to used more easily in scripts and loops.

- `rgb()` has a new 'maxColorValue' argument, allowing r,g,b in $[0, M]$, particularly in $0 : 255$, efficiently and non-error-prone.

- `summaryRprof()` provides the functionality of `R CMD Rprof` in R code, though more slowly.

- `stop()` accepts multiple arguments (which are concatenated) just as `warning()` does.

- `scan()` now throws an error with incorrect logical input (which was previously taken as `FALSE`).

- `pdf()` now uses PDF not R code for clipping, which ensures that partially visible text strings are (partially) shown.

- Each R session uses a per-session temporary directory which is removed at normal termination. The directory name is given by the `tempdir()` function, and filenames returned by `tempfile()` will be within that directory.

- `help.start()` on Unix now uses a '.R' subdirectory of the per-session temporary directory and not '`~/`.R'. A side effect is that '`~/`.R' is now never deleted by R.

  This now uses the remote control mechanism only if the X display is local to the R process (as otherwise it might use a browser running on an arbitrary machine).

- *Very* experimental `browseEnv()` for browsing objects in an environment.

- `cbind()` and `rbind()` used to ignore all zero-length vectors, an undocumented quirk for S-compatibility. This caused problems when combining zero-extent matrices and zero-length vectors, and now zero-length vectors are ignored unless the result would have zero rows/columns.

- `read.table(stdin())` will now work.

- `plot.spec(x)` now also works for other `x` than AR and Pgram results.

- New functions `La.chol()` and `La.chol2inv()` for Cholesky decomposition and inverse of positive definite matrices using Lapack.

- Changes to the **tcltk** package:

  – Added a few "trivial and obviously missing" functions: `tkchooseDirectory`, `tkpopup`, `tkdialog`, `tkread`.

  – on Unix systems, the Tcl event loop has been integrated with R's own (so that `tkwait.variable()` no longer halts updates of plot windows).

  – also on Unix, stubs have been created to divert R's input and output routines to go via Tcl commands. (Nothing uses this at present, but packages might be developed to take advantage of it.)

  – return value from Tcl commands is no longer invisible. A new print method, `print.tclObj()`, has been introduced.

  – Tcl variables created by `tclVar()` are now explicitly put into Tcl's global namespace, removing potential scoping problems.

  – The tcltk dynamic library now loads with `local=FALSE` since the default had trouble when loading Tcl extensions (e.g., Tix).

  – The `tkpager()` function had not been updated for the return value change from 1.5.0.

- The `bmp()`, `jpeg()` and `png()` devices can produce multiple bitmap files, one for each page. The default filenames have been changed to include a sequence number.

- New function `axTicks()` returning tick mark locations like `axis()`.

- `grid()` has a more sensible default behavior. Tick axis alignment only happens when no numbers of grid cells are specified. New arguments 'lwd' and 'equilogs'; nx/ny = NA for not drawing, see ?grid.

- `installed.packages()` has a new argument 'priority'.

- `termplot()` uses factor levels rather than $1, 2, 3, \ldots$ for x-axis.

- Workaround for optimization bugs on gcc 3.1/2 on 32-bit Solaris.

- The `trace()` function has been robustified and a new function `tracingState()` added to turn tracing temporarily on and off.

- New `cophenetic()` in **mva** as utility for hierarchical clustering.

- `p.adjust()` has two new methods, 'Hommel' and 'FDR', contributed by Gordon Smyth <smyth@wehi.edu.au>.

- `stars()` now has add and plot arguments.

- Enhancements to mathematical annotation of plots:

  – expressions involving `dot(`*something*`)` now produce a dot accent above the *something* (initial patch from Ben Bolker).

  – within an expression, the symbol '`partialdiff`' is now converted to a partial differential symbol (greek delta).

- `smooth.spline()` has a new argument 'nknots' allowing to set the default number of knots (when 'all.knots = FALSE' as per default).

Build Issues

- Toplevel 'Makefile' was missing dependency of '`docs`' on '`R`' (causing parallel makes to go wrong).

- When building with recommended packages those were installed into the first path in `R_LIBS`, if the environment variable was present.

Deprecated & defunct

- The assignment operator '`_`' is deprecated: a warning is given once per R session.

- `machine()`, `Machine()` and `Platform()` are deprecated in favour of `.Platform$OS.type`, `.Machine` and `.Platform`.

- `arima0.diag()` (package **ts**) is defunct.

- `piechart()` is defunct.

- `print.ordered()` has been removed, so `print.factor()` is used.

- The global internal variables `.Dyn.libs` and `.lib.loc` are removed in favor of the internal functions `.dynLibs()` and `.libPaths()`.

- `restart()` is deprecated in preparation for proper exception handling. Use `try()`, as has long been recommended.

Documentation changes

- New `demo(persp)` containing some of the former `example(persp)` ones and more.

C-level facilities

- 'Rversion.h' is no longer automatically included by 'R.h'. Include it explicitly if you need it.

- New entry point `R_tmpnam` in 'R_ext/Utils.h'.

- The Unix event loop interface has been changed to facilitate integration with other loops. `R_checkActivity` and `R_runHandlers` should eventually replace `getSelectedHandler`.

Installation changes

- Perl 5.005 or newer is now required.

- `R CMD INSTALL` is now guaranteed to sort the R source files in ASCII order.

Utilities

- `R CMD check` now tests for mis-use on an installed or binary package, and sets `T` and `F` to `NULL` when running the examples.

- New function `SweaveSyntConv()` converts between Sweave file syntaxes. `RweaveLatex()` now gets its prompt from `options()` and uses the text width as linebreak cutoff for deparsing input statements.

See the file 'NEWS' in the R distribution for additional information on bug fixes.

Changes on CRAN

by Kurt Hornik

CRAN packages

CGIwithR Facilities for the use of R to write CGI scripts. By David Firth.

ISwR Data sets and scripts for text examples and exercises in P. Dalgaard (2002), "Introductory Statistics with R", Springer Verlag. By Peter Dalgaard.

KMsurv Data sets and functions for Klein and Moeschberger (1997), "Survival Analysis, Techniques for Censored and Truncated Data", Springer. Original by Klein and Moeschberger, modifications by Jun Yan.

MPV Data sets from the book "Introduction to Linear Regression Analysis" by D. C. Montgomery, E. A. Peck, and C. G. Vining, 2001, John Wiley and Sons. By W. J. Braun.

RColorBrewer The package provides palettes for drawing nice maps shaded according to a variable. By Erich Neuwirth.

SparseM Basic linear algebra for sparse matrices. By Roger Koenker and Pin Ng.

StatDataML Read and write StatDataML files, alpha implementation of the StatDataML proposal. By Torsten Hothorn, Friedrich Leisch, and David Meyer.

ape Ape provides functions for reading, writing, and plotting phylogenetic trees in parenthetic format (standard Newick format), analyses of comparative data in a phylogenetic framework, analyses of diversification and macroevolution, computing distances from allelic and nucleotide data, reading nucleotide sequences from GenBank via internet, and several tools such as Mantel's test, computation of minimum spanning tree, or the population parameter theta based on various approaches. By Emmanuel Paradis, Korbibian Strimmer, Julien Claude, Yvonnick Noel, and Ben Bolker.

deal Bayesian networks with continuous and/or discrete variables can be learned and compared from data. By Susanne Gammelgaard Bøttcher and Claus Dethlefsen.

geepack Generalized estimating equations solver for parameters in mean, scale, and correlation structures, through mean link, scale link, and correlation link. Can also handle clustered categorical responses. By Jun Yan.

haplo.score A suite of routines that can be used to compute score statistics to test associations between haplotypes and a wide variety of traits, including binary, ordinal, quantitative, and Poisson. These methods assume that all subjects are unrelated and that haplotypes are ambiguous (due to unknown linkage phase of the genetic markers). The methods provide several different global and haplotype-specific tests for association, as well as provide adjustment for non-genetic covariates and computation of simulation p-values (which may be needed for sparse data). By Charles M. Rowland, David E. Tines, and Daniel J. Schaid. R version translation by Gregory R. Warnes.

mclust1998 Model-based cluster analysis: the 1998 version of MCLUST. By C. Fraley and A. E. Raftery, Dept. of Statistics, University of Washington. R port by Ron Wehrens.

msm Functions for fitting continuous-time Markov multi-state models to categorical processes observed at arbitrary times, optionally with misclassified responses, and covariates on transition or misclassification rates. By Christopher H. Jackson.

multcomp Multiple comparison procedures for the one-way layout. By Frank Bretz, Torsten Hothorn and Peter Westfall.

normix Onedimensional Normal Mixture Models Classes, for, e.g., density estimation or clustering algorithms research and teaching; providing the widely used Marron-Wand densities. By Martin Mächler.

qvcalc Functions to compute quasi-variances and associated measures of approximation error. By David Firth.

relimp Functions to facilitate inference on the relative importance of predictors in a linear or generalized linear model. By David Firth.

rgenoud A genetic algorithm plus derivative optimizer. By Walter R. Mebane, Jr., and Jasjeet Singh Sekhon.

sound Basic functions for dealing with '.wav' files and sound samples. By Matthias Heymann.

survrec Estimation of survival function for recurrent event data using Peña-Strawderman-Hollander, Whang-Chang estimators and MLE estimation under a Gamma Frailty model. Fortran 77 original by Edsel A Peña and Robert L Strawderman. Added Fortran routines, R code and packaging by Juan R González.

CRAN mirrors the R packages from the Omegahat project in directory 'src/contrib/Omegahat'. The following are recent additions:

RGtkBindingGenerator A meta-package which generates C and R code to provide bindings to a Gtk-based library. By Duncan Temple Lang.

RXLisp An interface to call XLisp-Stat functions from within R, inspired by Forrest Young's remarks about dynamic graphics, XLisp-Stat and R on R-devel. By Duncan Temple Lang.

Kurt Hornik
Wirtschaftsuniversität Wien, Austria
Technische Universität Wien, Austria
Kurt.Hornik@R-project.org

New Publications

The manual "An Introduction to R" by W. N. Venables, D. M. Smith and the R Development Core Team has now been published as a printed book (ISBN 0-9541617-4-2). It is available worldwide through the major online bookstores and wholesalers, at a retail price of $19.95 (£12.95 in UK). Further details are available at the publisher's website www.network-theory.co.uk.

gRaphical Models in R

A new initiative within the R project

Steffen L. Lauritzen

What is this?

In September 2002 a small group of people gathered in Vienna for the brainstorming workshop gR 2002 with the purpose of initiating the development of facilities in R for graphical modelling. This was made in response to the facts that:

- graphical models have now been around for a long time and have shown to have a wide range of potential applications
- software for graphical models is currently only available in a large number of specialised packages, such as BUGS, CoCo, DIGRAM, MIM, TETRAD, and others.

The time has come to integrate such facilities in general software, such as R, with flexible extension and modification of prepackaged modules. The workshop web page can be found on http://www.ci.tuwien.ac.at/Conferences/gR-2002/.

Summary of workshop

Two rather separate clusters of activities could be identified. In one, model selection and identification based on i.i.d. repetitions was the main issue and in the second, the primary issue was modularity in modelling and computation for complex patterns of observations.

Although some effort would be needed to accomodate both of these aspects, the aim of the initiative is to do so and get software from both of these clusters into R.

Further research is needed to make R move from computing within models to computing directly with 'abstract' models as objects, which seems necessary to represent the natural modularity of graphical models.

It was decided to take the following simple steps immediately:

WWW A web-page for the project has been set up at http://www.R-project.org/gR/.

SIG A special interest group for the gR project was formed with the associated mailing list R-sig-gR@lists.r-project.org. See the gR project web-page for subscription information.

DSC 2003 A session at the Distributed Statistical Computing workshop, taking place in Vienna in the period 20-22 March 2003, will be devoted to gRaphical models and the gR project.

Software Some software for graphical models was already integrated or is easily integrable in R and these packages would as quickly as possible be made available through CRAN. This includes

- **deal** for learning Bayesian networks (C. Dethlefsen and S. G. Bøttcher) is already available as an R package;
- the extensive program CoCo for analysis of discrete data (J. H. Badsberg);
- an interface making it possible to access MIM within R (S. Højsgaard);
- GRAPPA, a suite of R functions for probability propagation (P. Green).

gR 2003 A larger workshop is tentatively planned in Aalborg, Denmark, in September 2003.

Graph computations A special interest group had already been formed with the purpose of creating a module under R for computation with graphs. Such a module will be extremely valuable for the gR project.

Organisation Kurt Hornik will act as main contact between gR and the R Core team, and Claus Dethlefsen, Aalborg University, will serve as the maintainer of the CRAN view for R.

Steffen L. Lauritzen
Aalborg University, Denmark
steffen@math.auc.dk

Recent and Upcoming Events

R at the ICPSR summer program

R played a prominent role at the 2002 ICPSR Summer Program. Headquartered at the University of Michigan, the Inter-University Consortium for Political and Social Research (ICPSR) is an international organization of more than 400 colleges and universities. The Consortium sponsors a variety of services and activities, including an extensive social-science data archive and a highly regarded Summer Program in Quantitative Methods of Social Research.

The eight-week 40th edition of the ICPSR Summer Program attracted more than 700 participants—mostly graduate students and faculty in the social sciences—to Ann Arbor, Michigan to attend 30 courses. These courses ranged from the elementary to the advanced, most presented in intensive one-week, all-day classes, and in four-week, two-hours-per-day classes. Additionally, participants attended lectures on a variety of statistical topics. More information, including course outlines, is available at the ICPSR web site, http://www.icpsr.umich.edu/.

Computing in the ICPSR Summer Program has traditionally been eclectic, employing a wide range of statistical software. This year, several relatively advanced courses coordinated their use of R. In support of these courses and to gain more exposure for R among social scientists, I taught a two-week, two-hours-per-evening lecture on Statistical Computing in S, which featured R. More than 100 participants attended these lectures. R was installed in the ICPSR Windows-based computer labs, and a CD/ROM with R for Windows and Windows binaries for all of the packages on CRAN was made available to participants.

Four-week courses that employed R included Bayesian Methods for Social and Behavioral Sciences, taught by Jeff Gill of the University of Florida; Linear, Nonlinear, and Nonparametric Regression, taught by Bob Andersen, then of Oxford University, now of the University of Western Ontario; and Maximum Likelihood Estimation for Generalized Linear Models, taught by Charles Franklin of the University of Wisconsin. The combined enrollment of these classes was more than 100. In addition, Bill Jacoby of the University of South Carolina gave several lectures on statistical graphics and data visualization which featured R.

John Fox
McMaster University, Canada
jfox@mcmaster.ca

R featured at JSM

A good time was had at the Joint Statistical Meetings (JSM) in New York City, August 11–15, 2002. After several years of dismal locations (who can forget Dallas?), this year's JSM began a series of venues possibly even more interesting than the Meetings: in the next two years, the JSM will be in San Francisco and Toronto.

R was featured prominently at the JSM this year. In session 190, "The future of electronic publication: Show me ALL the data," organized by Brian Yandell and chaired by David Scott, a number of members of the R core team discussed R and related tools. Friedrich Leisch described StatDataML, an XML-based markup language for statistical data for the more easy transfer of data between programs, and Sweave, a system for creating statistical reports that combine text and code such that data analysis output can be created and inserted on the fly. Robert Gentleman emphasized that all statistical papers should be accompanied by sufficient data and software so that the results may be reproduced, a concept he called a "compendium." Such a compendium could be created as an R package containing a Sweave document. The key issue will likely be in distribution. Kurt Hornik described his experience in managing the R repository, which now contains over 165 packages. Central to the proper curation of such a software repository is the development of tools for automated testing, especially as the core system is updated. Duncan Temple Lang described a system for distributing verifiable, self-contained, annotated computations with interactive facilities so that readers may examine and explore the content on their own. The session concluded with a discussion by James Landwehr concerning the status of the electronic publication of the ASA's journals.

Session 349, "R Graphics," organized and chaired by Paul Murrell, included discussions of an R interface to OpenGL, the **scatterplot3d** package, and the integration of R graphics within Excel. In addition, Deborah Swayne demonstrated the GGobi data visualization system, a more modern version of xgobi, which allows multiple, linked graphical displays.

By my count, at least 22 of the technical sessions at the JSM included some discussion of the analysis of data from gene expression microarrays. The importance of the Bioconductor project (initiated by Robert Gentleman, and consisting largely of R packages for the analysis of microarray data) for statisticians working in the area was made clear. In particular, the R package **affy** (maintained by Rafael Irizarry), for the analysis of data from Affymetrix chips, was frequently mentioned.

Karl W. Broman
Johns Hopkins University
kbroman@jhsph.edu

DSC 2003

The third international workshop on *Distributed Statistical Computing (DSC 2003)* will take place at the Technische Universität Wien in Vienna, Austria from 2003-03-20 to 2003-03-22. This workshop will deal with future directions in (open source) statistical computing and graphics.

Topics of particular interest include

- Bioinformatics

- Database Connectivity
- Graphical Modeling
- GUIs and Office Integration
- Resample and Combine Methods
- Spatial Statistics
- Visualization

Emphasis will be given to the R (http://www.R-project.org/), Omegahat (http://www.omegahat.org/), and BioConductor (http://www.bioconductor.org/) projects. DSC 2003 builds on the spirit and success of DSC 1999 and 2001, which were seminal to the further development of R and Omegahat.

Deadline for registration is 2003-03-14. There will be a conference fee of EUR 200 for 'early' registrations made before 2003-02-14, and EUR 250 for registrations made afterwards.

On 2003-03-19 there will be several half-day tutorials, with topics currently including

- An Introduction to BioConductor
- Exploring Genomic Data using R and BioConductor
- R Graphics
- Writing R Extensions

Fees for each tutorial are EUR 50 (academic) or EUR 250 (non-academic).

Please contact the organizing committee at dsc-org@ci.tuwien.ac.at for further information.

Friedrich Leisch
Universität Wien
leisch@R-project.org

Computational and Statistical Aspects of Microarray Analysis

This one week intensive school is intended to give a clear view of current statistical and computational problems linked to microarray data along with some solutions. This self-contained course will touch on many aspects of genome biology as it applies to microarray analysis. Topics include preprocessing, estimating gene expression levels, microarray data and hybridization, experimental design, dimension reduction and pattern recognition techniques including boosting, bagging and other recent statistical techniques for microarray data analysis.

The course is primarily intended for PhD students and researchers in the areas of Statistics, Biology and related fields. A small background on data analysis is required. The course is computationally intensive and laboratory sessions are associated with methodology ones.

It will take place at the University of Milan, Italy, from 2003-05-26 to 2003-05-30 with two morning sessions on methodology and computer labs in the afternoon of each day. The course will be given by Anestis Antoniadis (Universite Joseph Fourier, Grenoble, France) and Robert Gentleman (Harvard School of Public Health, Boston, USA). Further information is available at http://www.eco-dip.unimi.it/marray.

Stefano Iacus
University of Milan
stefano.iacus@unimi.it